2025

Olimpiada Națională de Informatică pentru gimnaziu

Etapa județeană și etapa națională

Baraje de selecție a lotului național de juniori și a echipelor reprezentative ale României

Organizate în parteneriat de

SEPI
Societatea pentru Excelență și Performanță în Informatică

și Ministerul Educației și Cercetării

Infobits Academy

2025
Olimpiada Națională de Informatică pentru gimnaziu
Etapa județeană și etapa națională
Baraje de selecție a lotului național de juniori
și a echipelor reprezentative ale României

Enunțuri, soluții, surse

Tehnoredactare

Emanuela Cerchez, Cătălin Frâncu

Coperta

Emanuela Cerchez (pe baza unui design generat de Chat GPT)

ISBN

978-630-6559-21-3

Această lucrare este o **resursă educațională deschisă**, oferită gratuit tuturor celor care aspiră la excelență în informatică.

Societatea pentru Excelență și Performanță în Informatică
E-mail: contact@sepi.ro www.sepi.ro

Editura L&S SOFT
Telefon: 0727.731.947
E-mail: hello@infobits.ro
www.infobits.ro
ebooks.infobits.ro
Compania noastră oferă de peste 30 de ani manuale școlare aprobate de Ministerul Educației și auxiliare ce respectă programa școlară, aplicații online, precum și cursuri de Informatică și T.I.C., utile oricărei persoane care dorește să se pregătească în aceste domenii.

Autori

Comisia centrală a Olimpiadei Naționale de Informatică 2025, secțiunea Gimnaziu

Etapa județeană și etapa națională

Clasa a V-a

- prof. Adrian-Doru Pintea, Inspectoratul Școlar Județean Cluj - coordonator
- prof. Arnold Beiland, Liceul Teoretic Carei
- prof. Dan Octavian Dumitrașcu, Colegiul Național „Dinicu Golescu" Câmpulung
- prof. Eugenia Cristina Iordaiche, Liceul Teoretic „Grigore Moisil" Timișoara
- prof. Marius Nicoli, Colegiul Național „Frații Buzești" Craiova
- prof. Nicoleta Lenuța Șandor, Colegiul Național „Mihai Eminescu" Satu Mare
- prof. Roxana Gabriela Tîmplaru, Colegiul „Ștefan Odobleja" Craiova
- prog. Dan-Constantin Spătărel, SC Spătărel Tutoring SRL București
- stud. Jonathan Mogovan, Universitatea „Babeș-Bolyai", Cluj-Napoca

Clasa a VI-a

- prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj - coordonator
- prof. Ana-Maria Arișanu, Colegiul Național „Mircea cel Bătrân", Râmnicu-Vâlcea
- prof. Alice Georgescu, Colegiul Național „Mihai Viteazul", Ploiești
- prof. Alina Pintescu, Colegiul Național „Gheorghe Șincai", Baia Mare
- prof. Petru-Simion Opriță, Liceul „Regina Maria", Dorohoi
- prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian", Drobeta-Turnu Severin
- prof. Dan Pracsiu, Liceul Teoretic „Emil Racoviță", Vaslui
- prof. Marinel-Paul Șerban, Colegiul Național „Emil Racoviță", Iași
- stud. Petruț-Rareș Gheorghieș, Facultatea de Automatică și Calculatoare București
- stud. Alin-Gabriel Răileanu, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iași

Clasa a VII-a

- prof. Veronica-Raluca Costineanu, Colegiul Național „Ștefan cel Mare" Suceava - coordonator
- prof. Alina Gabriela Boca, Colegiul Național de Informatică „Tudor Vianu" București
- stud. Rareș-Andrei Cotoi, Facultatea de Matematică și Informatică, Universitatea Babeș-Bolyai Cluj-Napoca
- prof. Claudiu-Cristian Gorea-Zamfir, Inspectoratul Școlar Județean Iași
- stud. Mihai Marcu, Delft University of Technology
- prof. Vlad-Laurențiu Nicu, Liceul Teoretic „Mihail Kogălniceanu" Vaslui
- prof. Adrian Panaete, Colegiul Național „August Treboniu Laurian" Botoșani
- prof. Daniel Popa, Colegiul Național „Aurel Vlaicu" Orăștie
- stud. Ioan-Cristian Pop, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- prog. Cezar Trișcă-Vicol, 2k Games Dublin

Clasa a VIII-a

- prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță" Iași - coordonator

- stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București
- stud. Andrei Boacă, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” Iași
- prof. Isabela Patricia Coman, Colegiul Național de Informatică „Tudor Vianu” București
- stud. Victor Botnaru, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- stud. Răzvan Alexandru Rotaru, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” Iași
- Stud. Giulian Buzatu, Facultatea de Matematică-Informatică, Universitatea București
- prof. Nistor Moț, Liceul Teoretic „Dr. Luca” Brăila
- prof. Alin Burța, Colegiul Național „B. P. Hașdeu” Buzău
- prof. Florentina Ungureanu, Colegiul Național de Informatică Piatra Neamț

Barajul de selecție a lotului național de juniori

- prof. Adrian Panaete, Colegiul Național „A.T. Laurian” Botoșani - coordonator
- prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași
- stud. Alin Răileanu, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
- stud. Victor Botnaru, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian, Drobeta Turnu Severin
- stud. Răzvan Alexandru Rotaru, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” Iași
- stud. Rareș-Andrei Cotoi, Universitatea Babeș-Bolyai, Cluj, Facultatea de Matematică și Informatică
- stud. Giulian Buzatu, Facultatea de Matematică-Informatică, Universitatea București
- prof. Dan Pracsiu, Liceul Teoretic Emil Racoviță, Vaslui
- prof. Marinel Șerban, Colegiul Național „Emil Racoviță” Iași
- stud. Petruț-Rareș Gheorghieș, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- stud. Ioan-Cristian Pop, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București

Taberele de pregătire a lotului național de juniori și de selecție a echipelor reprezentative ale României pentru competițiile internaționale Craiova 9-14 mai 2025, Zalău 22-27 mai 2025

- prof. Adrian Panaete, Colegiul Național „A.T. Laurian” Botoșani - coordonator
- prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil” Buzău
- instr. Cristian Frâncu, Nerdvana București
- instr. Cătălin Frâncu, Nerdvana București
- stud. Andrei Boacă, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” Iași
- prof. Mihai Bunget, Colegiul Național Tudor Vladimirescu, Târgu-Jiu
- prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj, Târgu-Jiu
- prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași
- stud. Alin Răileanu, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” Iași
- stud. Victor Botnaru, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian, Drobeta Turnu Severin
- stud. Răzvan Alexandru Rotaru, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza” Iași

- stud. Rareș-Andrei Cotoi, Universitatea Babeș-Bolyai, Cluj, Facultatea de Matematică și Informatică
- stud. Giulian Buzatu, Facultatea de Matematică-Informatică, Universitatea București
- prof. Dan Pracsiu, Liceul Teoretic Emil Racoviță Vaslui
- prof. Marinel Șerban, Colegiul Național „Emil Racoviță" Iași
- stud. Petruț-Rareș Gheorghieș, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- stud. Ioan-Cristian Pop, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București

Cuprins

Partea I

Olimpiada Națională de Informatică
- etapa județeană -

16 martie 2025

Capitolul 1

OJI 2025, clasa a V-a

1.1 Problema Palindrom

Propusă de: Dan-Constantin Spătărel, București

Oglinditul unui număr natural este obținut din cifrele acestuia, citite de la dreapta la stânga. Un număr natural este **palindrom** dacă este egal cu oglinditul său. De exemplu, numărul 121 este palindrom deoarece oglinditul său este tot 121, iar numărul 124 nu este palindrom deoarece oglinditul său este 421.

Inserarea unei cifre într-un număr natural se poate face înainte de prima cifră a numărului (numai dacă cifra inserată este nenulă), după ultima cifră a numărului sau între oricare două cifre învecinate.

Se dă un număr natural N și apoi N numere naturale, toate având același număr de cifre.

Cerințe

1. Determinați câte dintre cele N numere sunt palindrom.
2. Determinați câte dintre cele N numere pot deveni palindrom prin inserarea în acestea a câte unei cifre.
3. Determinați câte dintre cele N numere pot deveni palindrom prin inserarea în acestea a câte două cifre.

Date de intrare

Fișierul de intrare `palindrom.in` conține:

- pe prima linie un număr natural C, reprezentând numărul cerinței, care poate avea valorile 1, 2 sau 3;
- pe a doua linie un număr natural N, având semnificația din enunț;
- pe a treia linie N numere naturale, despărțite prin câte un spațiu, având semnificația din enunț.

Date de ieșire

În fișierul de ieșire `palindrom.out` se afișează, pe prima linie, un număr natural reprezentând rezultatul determinat conform cerinței C.

Restricții

- $1 \leq N \leq 100\,000$
- Toate numerele de pe a treia linie au același număr de cifre, notat cu X
- $2 \leq X \leq 9$

#	Puncte	Restricții	
1	41	$C = 1$	
2	11	$C = 2$,	$X \leq 3$
3	18	$C = 2$,	$X > 3$
4	11	$C = 3$,	$X \leq 4$
5	19	$C = 3$,	$X > 4$

Exemple

palindrom.in	palindrom.out	Explicații
1 3 12321 10301 10331	2	12321 și 10301 sunt palindrom. 10331 nu este palindrom.
2 4 232 233 243 990	2	232 devine 2**3**32 prin inserarea unei cifre, care este palindrom. 233 devine 233**2** prin inserarea unei cifre, care este palindrom. 243 nu poate deveni palindrom prin inserarea unei singure cifre. 990 nu poate deveni palindrom prin inserarea unei singure cifre (nu se permite inserarea cifrei 0 înainte de prima cifră a numărului).
3 5 1221 1231 3112 9880 9890	4	1221 devine 12**33**21, care este palindrom. 1231 devine 12**3**3**2**1, care este palindrom. 3112 devine **2**3113**2**, care este palindrom. 9880 devine 9**0**880**9**, care este palindrom. 9890 nu poate deveni palindrom prin inserarea a două cifre.

1.2 Rezolvarea problemei Palindrom

Cerința 1

În continuare vom descrie rezolvarea pentru un singur număr. Pentru rezolvarea problemei se aplică algoritmul de mai jos în mod repetat, de N ori.

Pentru rezolvarea primei cerințe vom elimina simultan prima și ultima cifră a numărului cât timp acestea coincid și numărul are cel puțin două cifre. La final, dacă numărul rămas este 0 sau are o cifră, atunci putem afirma că numărul inițial era palindrom.

Cerința 2

Pentru rezolvarea celei de-a doua cerințe, observăm că operația de inserare a unei cifre în numărul dat și verificarea ulterioară a proprietății de palindrom este echivalentă cu operația de ștergere a unei cifre din numărul dat și verificarea proprietății de palindrom a numărului astfel obținut.

În plus, observăm că în orice număr palindrom putem insera o cifră (în mijlocul său) astfel încât numărul nou obținut să fie tot palindrom.

Deci, orice număr care respectă prima cerință o respectă și pe a doua.

Pentru a determina eficient care cifră ar trebui ștearsă, vom efectua următorul algoritm:

1. vom elimina simultan prima și ultima cifră a numărului cât timp acestea coincid și numărul are cel puțin două cifre;
2. dacă numărul rămas este 0 sau are o cifră, atunci putem spune că răspunsul la cerință este afirmativ;
3. altfel vom investiga două cazuri:
   (a) dacă eliminăm prima cifră;
   (b) dacă eliminăm ultima cifră;
4. în ambele cazuri, după eliminarea cifrei corespunzătoare, vom repeta pașii (1) și (2);
5. dacă la final numărul rămas are mai mult de o cifră, atunci răspunsul la cerință este negativ;

Cerința 3

Pentru rezolvarea celei de-a treia cerințe, observăm că operația de inserare a două cifre în numărul dat și verificarea ulterioară a proprietății de palindromicitate este echivalentă cu operația de ștergere a două cifre din numărul dat și verificarea proprietății de palindromicitate a numărului astfel obținut.

În plus, observăm că în orice număr palindrom putem insera o cifră (în mijlocul său) astfel încât numărul nou obținut să fie tot palindrom.

Deci, orice număr care respectă a doua cerință o respectă și pe a treia.

Pentru a determina eficient care cifre ar trebui șterse, vom efectua următorul algoritm:

(A) pentru ștergerea primei cifre, vom efectua pașii (1), (2) și (3);
(B) pentru ștergerea celei de-a doua cifre, vom efectua din nou pașii (1), (2) și (3);
(C) pentru determinarea răspunsului, vom efectua iarăși pașii (1), (2) și (5).

Observați că, efectuând de două ori pasul (3), vom ajunge să investigăm 4 scenarii:

1. la prima nepotrivire eliminăm prima cifră iar la a doua nepotrivire eliminăm din nou prima cifră;
2. la prima nepotrivire eliminăm prima cifră iar la a doua nepotrivire eliminăm a doua cifră;
3. la prima nepotrivire eliminăm a doua cifră iar la a seconda nepotrivire eliminăm prima cifră;
4. la prima nepotrivire eliminăm a doua cifră iar la a doua nepotrivire eliminăm din nou a doua cifră.

Pentru a evita cazul particular în care nu avem voie să adăugăm cifra 0 la începutul numărului inițial, atunci când eliminăm ultima cifră trebuie să verificăm și să ignorăm cazul în care numărul rămas pe care lucrăm este numărul original și ultima sa cifră este 0.

Complexitatea timp: $O(N \cdot X)$

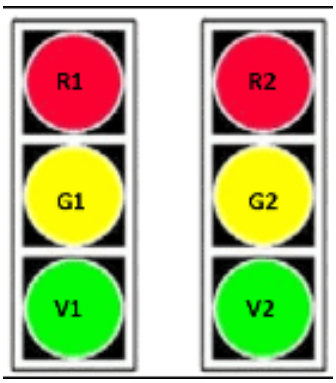
1.3 Cod-sursă pentru problema Palindrom

```cpp
#include <fstream>

int main() {
  std::ifstream fisier_in("palindrom.in");
  std::ofstream fisier_out("palindrom.out");
  int C, N;
  fisier_in >> C >> N;
  int raspuns = 0;
  for (int i = 0; i < N; i++) {
    int nr;
    fisier_in >> nr;
    bool este_bun = false;
    int nr_original = nr;
    // Calculez puterea lui 10 care are la fel de multe cifre ca și numărul citit.
    // Mă va ajuta să aflu care este prima cifră a numărului și eventual să o elimin.
    int pow10 = 1;
    while (nr / pow10 > 9) {
      pow10 *= 10;
    }
    // Cât timp numărul are cel puțin două cifre iar
    // prima și ultima cifră a numărului coincid, le elimin.
    while (pow10 > 1 && nr / pow10 == nr % 10) {
      nr %= pow10;
      nr /= 10;
      pow10 /= 100;
    }
    if (pow10 <= 1) {
      // Dacă numărul rămas este 0 sau are o singură cifră, atunci
      // indiferent de cerința pe care trebuie să o rezolv, răspunsul este afirmativ.
      este_bun = true;
    } else if (C >= 2) {
      // Dacă cerința este 2 sau 3, trebuie să investighez două scenarii:
      // (dacă op1 == 1) din numărul rămas voi elimina prima cifră;
      // (dacă op1 == 2) din numărul rămas voi elimina ultima cifră.
      for (int op1 = 1; op1 <= 2; op1++) {
        // Voi lucra pe o copie a numărului.
        int copie1_nr = nr;
        if (op1 == 1) {
          copie1_nr %= pow10;
        } else {
          // Dacă încerc să elimin ultima cifră din numărul original
          // și aceasta este 0, atunci înseamnă că ceea ce fac este
          // echivalent cu a insera o cifră de 0 în fața numărului
          // original, ceea ce nu este permis.
          if (copie1_nr == nr_original && copie1_nr % 10 == 0) {
            continue;
          }
          copie1_nr /= 10;
        }
        int copie1_pow10 = pow10 / 10;
        // Cât timp numărul are cel puțin două cifre iar
        // prima și ultima cifră a numărului coincid, le elimin.
        while (copie1_pow10 > 1 && copie1_nr / copie1_pow10 == copie1_nr % 10) {
          copie1_nr %= copie1_pow10;
          copie1_nr /= 10;
          copie1_pow10 /= 100;
        }
        if (copie1_pow10 <= 1) {
          // Dacă numărul rămas este 0 sau are o singură cifră, atunci
```

```
          // indiferent de cerința pe care trebuie să o rezolv, răspunsul este afirmativ.
          este_bun = true;
        } else if (C == 3) {
          // Dacă cerința este 3, trebuie să investighez două scenarii:
          // (dacă op2 == 1) din numărul rămas voi elimina prima cifră;
          // (dacă op2 == 2) din numărul rămas voi elimina ultima cifră.
          for (int op2 = 1; op2 <= 2; op2++) {
            // Voi lucra pe o a doua copie a numărului.
            int copie2_nr = copie1_nr;
            if (op2 == 1) {
              copie2_nr %= copie1_pow10;
            } else {
              copie2_nr /= 10;
            }
            int copie2_pow10 = copie1_pow10 / 10;
            // Cât timp numărul are cel puțin două cifre iar
            // prima și ultima cifră a numărului coincid, le elimin.
            while (copie2_pow10 > 1 && copie2_nr / copie2_pow10 == copie2_nr % 10) {
              copie2_nr %= copie2_pow10;
              copie2_nr /= 10;
              copie2_pow10 /= 100;
            }
            // Trebuie să rezolv cerința 2 și verific dacă am rămas
            // cu un număr de o singură cifră sau cu 0.
            if (copie2_pow10 <= 1) {
              este_bun = true;
            }
          }
        }
      }
    }
    if (este_bun) {
      raspuns++;
      //fisier_out << nr_original;
    }
  }
  fisier_out << raspuns;
  return 0;
}
```

1.4 Problema Semafoare

Propusă de: prof. Cristina Iordaiche, Liceul Teoretic „Grigore Moisil" Timișoara



Un dispozitiv de tip semafor are trei culori, roșu, galben și verde, și funcționează ciclic, astfel încât, în fiecare moment, să fie aprinsă o singură culoare. Într-o serie, culorile se succed întotdeauna în ordinea următoare: roșu, galben, verde, galben. Astfel, la pornire se aprinde roșu, iar după ce se stinge această culoare se aprinde galben, apoi verde și apoi, din nou, galben, apoi seria culorilor se reia ciclic, în succesiunea precizată. Pentru două semafoare se testează acum modul de funcționare. La primul semafor, într-o serie roșu stă aprins R1 secunde, apoi se aprinde galben, pentru G1 secunde, apoi se aprinde verde, pentru V1 secunde, apoi din nou galben, pentru G1 secunde. La al doilea semafor, într-o serie roșu stă aprins R2 secunde, apoi se aprinde galben, pentru G2 secunde, apoi se aprinde verde, pentru V2 secunde, și din nou galben, pentru G2 secunde.

În acest moment, au trecut T1 secunde de la pornirea primului semafor și T2 secunde de la pornirea celui de-al doilea semafor.

Cerințe

1. Știind că în acest moment la niciunul dintre semafoare nu este aprins verde, determinați numărul minim de secunde care trebuie să treacă, din acest moment, până când se aprinde verde la cel puțin unul dintre ele.
2. Determinați numărul minim de secunde care trebuie să treacă, din acest moment, până când ambele semafoare au aprinsă aceeași culoare.

Date de intrare

Fișierul de intrare `semafoare.in` conține:

- pe prima linie, un număr natural, C, reprezentând numărul cerinței (1 sau 2);
- pe a doua linie, trei numere naturale, R1, G1, V1, în această ordine, cu semnificația din enunț;
- pe a treia linie, trei numere naturale, R2, G2, V2, în această ordine, cu semnificația din enunț;
- pe a patra linie, două numere naturale, T1 și T2, în această ordine, cu semnificația din enunț.

Numerele aflate pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `semafoare.out` conține, pe prima linie, un număr natural, reprezentând rezultatul determinat conform cerinței C.

Restricții

- $R1$, $G1$, $V1$, $R2$, $G2$, $V2$ sunt numere naturale nenule, cu cel mult 5 cifre fiecare
- $0 \leq T1, T2 \leq 1\,000\,000\,000$
- Pentru datele furnizate, se garantează că există întotdeauna soluție

#	Puncte	Restricții	
1	33	$C = 1$,	$T1 = 0$ și $T2 = 0$
2	35	$C = 1$,	$T1 + T2 > 0$
3	13	$C = 2$,	$1 \leq T1, T2 \leq 100\,000$
4	19	$C = 2$,	$100\,001 \leq T1, T2 \leq 1\,000\,000\,000$

Exemple

semafoare.in	semafoare.out	Explicații
1 2 4 2 3 1 3 0 0	4	Primul semafor pornește în acest moment cu roșu, care stă aprins 2 secunde, apoi galben 4 secunde și verde 2 secunde. Trec $2 + 4 = 6$ secunde până când se aprinde verde. Al doilea semafor pornește în acest moment cu roșu, care stă aprins 3 secunde, galben 1 secundă și verde 3 secunde. Trec $3 + 1 = 4$ secunde până când se aprinde verde. Numărul minim de secunde care trebuie să treacă din acest moment până când se aprinde verde la unul dintre semafoare este egal cu 4.
1 2 4 2 3 1 3 4 1	2	Primul semafor a pornit de 4 secunde, deci în acest moment este deja aprins galben, de 2 secunde, iar peste 2 secunde urmează verde. Al doilea semafor a pornit de 1 secundă, deci în acest moment este deja aprins roșu, de 1 secundă, iar peste 2 secunde urmează galben, apoi peste încă o secundă urmează verde (în total peste 3 secunde). Numărul minim de secunde care trebuie să treacă din acest moment până când se aprinde verde la unul dintre semafoare este egal cu 2.
2 2 4 2 3 1 3 3 2	1	Primul semafor a pornit de 3 secunde, deci în acest moment este deja aprins galben, de 1 secundă, care stă aprins încă 3 secunde. Al doilea semafor a pornit de 2 secunde, deci în acest moment este deja aprins roșu, de 2 secunde, iar peste 1 secundă urmează galben. După o secundă din acest moment este aprins galben, la ambele semafoare.

1.5 Rezolvarea problemei Semafoare

Cerința 1

Pentru cazurile în care $T_1 = 0$ și $T_2 = 0$, se poate calcula, cu o formulă simplă, după câte secunde se face verde la unul dintre cele două semafoare:

- Calculăm pentru fiecare semafor totalul secundelor care trebuie să treacă până când se aprinde galben după roșu, iar apoi verde după galben.
- Afișăm timpul minim astfel calculat.

Algoritm:

- calculăm totalul de secunde necesare fiecărui semafor
- afișăm timpul minim calculat:

---

if $R_1 + G_1 < R_2 + G_2$ **then**
  afișează $R_1 + G_1$
else
  afișează $R_2 + G_2$

Pentru cazurile în care $T_1 + T_2 > 0$ (cel puțin unul dintre cele două semafoare nu pornește la momentul curent), observăm că fiecare semafor funcționează pe baza unui ciclu temporar ce se repetă continuu. O soluție posibilă constă în parcurgerea următorilor pași:

- calculăm durata ciclului pentru fiecare semafor:

$ciclu1 \leftarrow R_1 + G_1 + V_1 + G_1$
$ciclu2 \leftarrow R_2 + G_2 + V_2 + G_2$

- calculăm poziția în ciclul fiecărui semafor:

$T_1 \leftarrow T_1 \mod ciclu1$
$T_2 \leftarrow T_2 \mod ciclu2$

- determinăm timpul până se aprinde verde la primul semafor:

---

if $T_1 < R_1 + G_1$ **then**
  $timp\_pana\_la\_verde1 \leftarrow R_1 + G_1 - T_1$
else if $T_1 < R_1 + G_1 + V_1$ **then**
  $timp\_pana\_la\_verde1 \leftarrow 0$
else
  $timp\_pana\_la\_verde1 \leftarrow ciclu1 - T_1 + R_1 + G_1$

- determinăm timpul până se aprinde verde la cel de-al doilea semafor (similar)
- determinăm care semafor ajunge primul pe verde:

---

if $timp\_pana\_la\_verde1 < timp\_pana\_la\_verde2$ **then**
  afișează $timp\_pana\_la\_verde1$
else
  afișează $timp\_pana\_la\_verde2$

Cerința 2

O soluție posibilă constă în parcurgerea următorilor pași:

- Calculăm durata totală a ciclului pentru fiecare semafor, similar cu cerința anterioară.
- Identificăm pentru fiecare semafor culoarea aprinsă la momentul curent. De exemplu, la momentul curent t_1, culoarea primului semafor se poate determina astfel:

---

if $t_1 < R_1$ **then**
  $culoare1 \leftarrow 0$ (Roșu)
else
  if $t_1 < R_1 + G_1$ **then**
    $culoare1 \leftarrow 1$ (Galben)
  else
    if $t_1 < R_1 + G_1 + V_1$ **then**
      $culoare1 \leftarrow 2$ (Verde)
    else
      $culoare1 \leftarrow 1$ (Galben)

- Simulăm scurgerea timpului, din secundă în secundă, până când la ambele semafoare se va observa aceeași culoare.

1.6 Cod-sursă pentru problema Semafoare

```
#include <fstream>
using namespace std;
ifstream fin("semafoare.in");
ofstream fout("semafoare.out");
int C;
int R1, G1, V1, R2, G2, V2, T1, T2;
int main()
{
  fin>>C;
  fin >> R1 >> G1 >> V1 >> R2 >> G2 >> V2 >> T1 >> T2;
  if(C==1 && T1==0 && T2==0)
    {
      if(R1+G1<R2+G2)
        fout<<(R1+G1)<<'\n';
      else fout<<(R2+G2)<<'\n';
    }
  int ciclu1 = R1 + G1 + V1 + G1;
  int ciclu2 = R2 + G2 + V2 + G2;
  T1 = T1 % ciclu1;
  T2 = T2 % ciclu2;
  if(C==1 && T1+T2>0)
    {
      int timp_la_verde1;
      if (T1 < R1 + G1)
        timp_la_verde1 = R1 + G1 - T1;
        else
            if (T1 < R1 + G1 + V1)
        timp_la_verde1 = 0;
      else
        timp_la_verde1 = ciclu1 - T1 + R1 + G1;

      int timp_la_verde2;
```

```
      if (T2 < R2 + G2)
        timp_la_verde2 = R2 + G2 - T2;
        else
            if (T2 < R2 + G2 + V2)
        timp_la_verde2 = 0;
      else
        timp_la_verde2 = ciclu2 - T2 + R2 + G2;

      if (timp_la_verde1 < timp_la_verde2)
        fout << timp_la_verde1 << '\n';
      else
        fout << timp_la_verde2 << '\n';
    }
  else if(C==2)
    {
      int culoare1, culoare2;
      int timp_minim = 0;
      bool gasit = false;
      while (!gasit)
        {
          // Calculăm culoarea pentru fiecare semafor la timpul curent
          int t1 = (T1 + timp_minim) % ciclu1;
          int t2 = (T2 + timp_minim) % ciclu2;
          // Determinăm culoarea semaforului 1
          if (t1 < R1)
            culoare1 = 0; // Rosu
            else
                if (t1 < R1 + G1)
            culoare1 = 1; // Galben
                else
                     if (t1 < R1 + G1 + V1)
            culoare1 = 2; // Verde
          else
            culoare1 = 1; // Galben
          // Determinăm culoarea semaforului 2
          if (t2 < R2)
            culoare2 = 0; // Rosu
            else
                if (t2 < R2 + G2)
            culoare2 = 1; // Galben
                else
                    if (t2 < R2 + G2 + V2)
            culoare2 = 2; // Verde
          else
            culoare2 = 1; // Galben
          // Verificăm dacă cele două semafoare au aceeași culoare
          if (culoare1 == culoare2)
            gasit = true;
          else
            timp_minim++;
        }
      fout << timp_minim << endl;
    }
  return 0;
}
```

Capitolul 2

OJI 2025, clasa a VI-a

2.1 Problema Avion

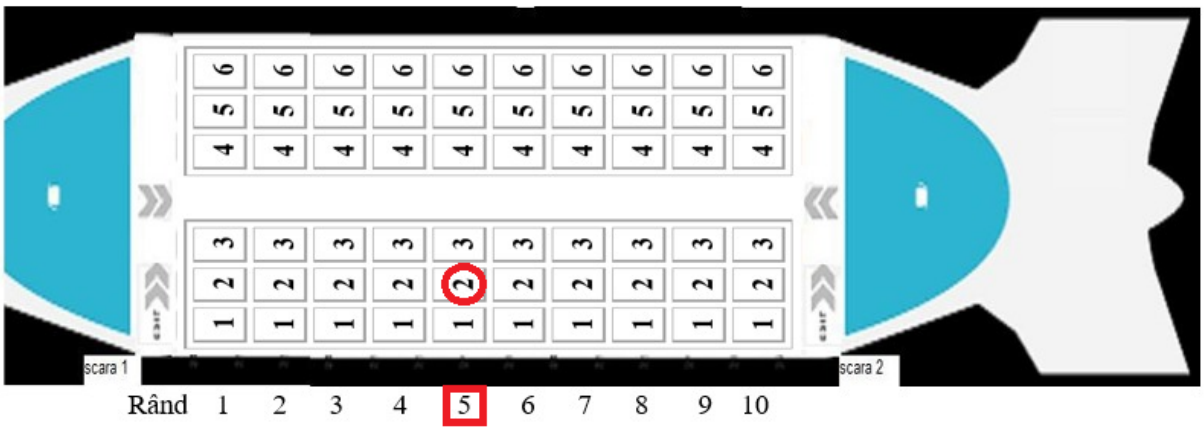
Propusă de: prof. Marinel-Paul Șerban, Colegiul Național „Emil Racoviță”, Iași

Avionul cu care am zburat ultima dată are o organizare foarte simplă. Pe fiecare rând sunt 6 scaune, câte 3 pe fiecare parte, având la mijloc culoarul pe care intră și ies pasagerii. Rândurile de scaune pentru pasageri sunt numerotate de la 1 la Nr, începând dinspre cabina piloților avionului. Pe fiecare rând, scaunele sunt numerotate cu cifre de la 1 la 6.

Urcarea în avion se face pe una dintre cele două scări: scara 1, situată în partea din față a avionului, și scara 2, situată în partea din spate a acestuia.

De la intrarea în avion fiecare pasager parcurge 3 metri până la culoarul principal, după care înaintează pe culoar parcurgând câte 1 metru pentru fiecare rând de scaune. De pe culoar până la scaunele 3 sau 4 se parcurge 1 metru, până la scaunele 2 sau 5 se parcurg 2 metri, iar până la scaunele 1 sau 6 se parcurg 3 metri.

În așteptare sunt n pasageri, care trebuie să urce în avion pe una din cele două scări. Pe biletul fiecărui pasager este scris locul pe care trebuie să îl ocupe în avion, sub forma perechii ***rând scaun*** (de exemplu 5 2 – rândul 5 scaunul 2).



Se cunosc numărul de rânduri de scaune Nr din avion, numărul n de pasageri și locul fiecărui pasager.

Cerințe

1. Determinați pentru fiecare dintre cei n pasageri, scara pe care trebuie să urce în avion, astfel încât distanța parcursă de el până la locul său să fie minimă.
2. Determinați distanța totală minimă parcursă de pasageri în avion. Distanța totală parcursă este egală cu suma distanțelor minime parcurse de cei n pasageri până la locurile lor.

Date de intrare

Fișierul de intrare **avion.in** conține pe prima linie trei numere naturale: c, reprezentând cerința care trebuie rezolvată ($c \in \{1, 2\}$), Nr și n, cu semnificațiile din enunț. Fiecare dintre următoarele n linii conține câte o pereche de numere naturale, reprezentând locul unui pasager, în ordinea în care aceștia stau în așteptare. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Pentru cerința 1 (dacă $c = 1$) fișierul de ieșire **avion.out** conține n linii, pe fiecare linie fiind cifra 1 sau cifra 2, reprezentând scara pe care urcă fiecare pasager, în ordinea în care aceștia au stat în așteptare.

Pentru cerința 2 (dacă $c = 2$) fișierul de ieșire **avion.out** conține un număr natural, reprezentând distanța totală minimă determinată la cerința 2.

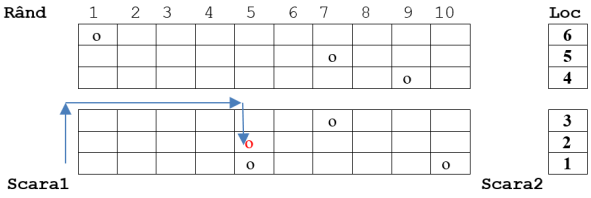
Restricții

- $c \in \{1, 2\}$
- $6 \leq Nr \leq 50$, Nr este număr par
- $1 \leq n \leq 6 \times Nr$
- Există zboruri în care nu sunt ocupate toate locurile

#	Puncte	Restricții
1	50	pentru cerința 1
2	50	pentru cerința 2

Exemple

avion.in	avion.out	Explicații
1 10 7 5 2 9 4 5 1 7 5 1 6 8 3 10 1	1 2 1 2 1 2 2	Se rezolvă cerința 1. Este un avion cu 10 rânduri de scaune și sunt 7 pasageri. Dacă ar urca pe scara 1, primul pasager ar parcurge $3 + 5 + 2$ metri, iar dacă ar urca pe scara 2 ar parcurge $3 + 6 + 2$ metri. Ca urmare, va alege să urce pe scara 1. Analog se procedează și pentru ceilalți pasageri. Astfel, pasagerii cu numerele de ordine 1, 3 și 5 urcă pe scara 1, iar pasagerii cu numerele de ordine 2, 4, 6 și 7 urcă pe scara 2.

2 10 7 5 2 9 4 5 1 7 5 1 6 8 3 10 1	57	Se rezolvă cerința 2. Este un avion cu 10 rânduri de scaune și sunt 7 pasageri. Pentru a parcurge distanța minimă până la locul său, pasagerul 1 urcă pe scara 1, parcurge 3 metri până la culoarul central apoi parcurge 5 metri pe culoar, apoi 2 metri până la scaunul alocat $(5 + 2)$. El parcurge astfel 10 metri până la locul său.  Pasagerul 2 urcă pe scara 2 și parcurge 6 metri $(3 + 2 + 1)$. Pasagerul 3 urcă pe scara 1 și parcurge 11 metri $(3 + 5 + 3)$. Pasagerul 4 urcă pe scara 2 și parcurge 9 metri $(3 + 4 + 2)$. Pasagerul 5 urcă pe scara 1 și parcurge 7 metri. Pasagerul 6 urcă pe scara 2 și parcurge 7 metri. Pasagerul 7 urcă pe scara 2 și parcurge 7 metri. În total au fost parcurși 57 de metri.

2.2 Rezolvarea problemei Avion

Cerința 1 - 50p

Având în vedere că numărul de rânduri Nr este par, există același număr de rânduri în fiecare jumătate a avionului.

Este suficient să fe comparat cu $Nr/2$ rândul de pe biletul fiecărui pasager. Dacă rândul este în prima jumătate a avionului (rândul $\leq Nr/2$) se va afișa 1, altfel se va afișa 2.

Cerința 2 - 50p

Pentru fiecare pasager se va compara și de această dată rândul de pe bilet cu mijlocul avionului $(Nr/2)$. Dacă rândul $\leq Nr/2$ și pasagerul urcă în avion pe scara 1, la suma totală se va adăuga rândul de pe bilet, iar în caz contrar, când pasagerul intră pe scara 2, el va parcurge până la rândul de pe bilet distanța $Nr - randul + 1$, care se va adăuga la suma totală. Apoi, pasagerul va mai parcurge 1, 2 sau 3 metri în funcție de locul ocupat pe rândul de pe bilet: 1 metru dacă locul este 3 sau 4, 2 metri dacă locul este 2 sau 5 și 3 metri dacă locul este 1 sau 6. Aceste valori se adaugă și ele la suma totală. Având în vedere că fiecare dintre cei n pasageri parcurge 3 metri de la intrarea în avion până la culoarul central, la suma totală se mai adaugă 3 * n metri.

2.3 Cod-sursă pentru problema Avion

```cpp
#include <bits/stdc++.h>

using namespace std;

ifstream fin("avion.in");
ofstream fout("avion.out");

int C, n, NR, i, randul, k, total_dist, loc, litera;

int main()
{
    fin >> C >> NR >> n;                          //citesc C, n si NR
    if (C == 1)                                   //daca cerinta este 1
    {
        for (i = 1; i <= n; i++)                  //citesc cele n locuri
        {
            fin >> randul >> loc;
            if (randul <= NR / 2)                 //e in prima jumatate a avionului?
                fout << 1 << '\n';                //afiseaza scara 1
            else
                fout << 2 << '\n';                //altfel afiseaza scara 2
        }
    }
    else                                          //cerinta 2
    {
        total_dist = 0;
        for (i = 1; i <= n; i++)                  //citesc cei n pasageri
        {
            fin >> randul >> loc;
            if (randul <= NR / 2)                 //e in prima jumatate a avionului?
                total_dist += randul;             //se aduna direct randul (pe culoar)
            else                                  //a urcat pe scara 2
                total_dist += (NR - randul + 1);  //parcurge pe culoar NR-randul+1
            if (loc == 1 || loc == 6)             //are loc la geam
                total_dist += 3;
            if (loc == 2 || loc == 5)             //are loc la mijloc
                total_dist += 2;
            if (loc == 3 || loc == 4)             //are loc langa culoar
                total_dist += 1;
        }
        fout << total_dist + 3 * n << '\n';       //se mai aduna 3 m pana la culoar
    }
    return 0;
}
```

2.4 Problema Mandatar

Propusă de: prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj

Se consideră șirul $A = (A_1, A_2, ..., A_n)$ cu n numere naturale nenule. Pe baza șirului A se construiește șirul B, unde fiecare element B_i este cel mai mic număr natural care are aceiași factori primi cu A_i, cu $1 \leq i \leq n$.

Exemplu: Dacă $A_1 = 24$, acesta se descompune în $2^3 \cdot 3^1$ și are factorii primi 2 și 3. Ca urmare, $B_1 = 6$ ($6 = 2^1 \cdot 3^1$) este cel mai mic număr natural care are aceiași factori primi cu 24.

O secvență de cel puțin două numere aflate pe poziții consecutive în șirul B este **mandatorie** dacă există un număr x ($2 \leq x \leq 9$) în această secvență care divide fiecare dintre elementele secvenței. Numim acest număr x - **mandatar** al secvenței. Lungimea secvenței este egală cu numărul de elemente ale acesteia.

Exemplu: secvența 6, 14, 2, 22, 2, 70 este o secvență mandatorie pentru că toate numerele care o compun sunt divizibile cu $x = 2$, număr cuprins între 2 și 9, ce aparține secvenței. Lungimea secvenței este 6.

Cerințe

1. Determinați cel mai mare număr prim din șirul A.
2. Determinați cel mai mare număr al șirului B ce are un număr maxim de factori primi.
3. Determinați lungimea maximă a unei secvențe mandatorii din șirul B.

Date de intrare

Fișierul de intrare **mandatar.in** conține pe prima linie numărul natural c, reprezentând cerința care trebuie rezolvată (1,2 sau 3), pe linia a doua numărul natural n, cu semnificația din enunț, iar pe următoarea linie n numere naturale, separate prin câte un spațiu, reprezentând elementele șirului A.

Date de ieșire

Fișierul de ieșire **mandatar.out** conține numărul determinat pentru cerința c.

Restricții

- $c \in \{1, 2, 3\}$
- $1 \leq n \leq 100\,000$
- $2 \leq A_i \leq 10^7$, $1 \leq i \leq n$
- $2 \leq x \leq 9$

#	Puncte	Restricții
1	50	$c = 1$. Șirul A conține cel puțin un număr prim.
2	30	$c = 2$.
3	20	$c = 3$. Șirul B conține cel puțin o secvență mandatorie.

Exemple

mandatar.in	mandatar.out	Explicații
1 10 17 45 9 90 66 24 2 40 29 4	29	$c = 1$, $n = 10$. Se rezolvă cerința 1. Dintre cele 10 elemente ale șirului A, numerele 17, 2, 29 sunt numere prime, iar numărul 29 este cel mai mare dintre acestea.
2 10 17 45 9 90 66 24 2 40 29 4	66	$c = 2$, $n = 10$. Se rezolvă cerința 2. Se construiește șirul B pe baza șirului A, după cum urmează: 17 15 3 30 66 6 2 10 29 2. Sunt două elemente care au număr maxim de factori primi (câte 3 factori primi): 30 și 66, iar 66 este cel mai mare.
3 10 17 45 9 90 66 24 2 40 29 4	5	$c = 3$, $n = 10$. Se rezolvă cerința 3. Se construiește șirul B pe baza șirului A, după cum urmează: 17 15 3 30 66 6 2 10 29 2. Sunt două secvențe mandatorii de lungime maximă, care este egală cu 5: 15 3 30 66 6; 30 66 6 2 10.

2.5 Rezolvarea problemei Mandatar

Cerința 1 - 50p

Se rețin numerele citite din fișierul de intrare în șirul A - tablou unidimensional (vector). Pentru determinarea celui mai mare număr prim din șirul A se folosește un algoritm de verificare a primalității unui număr.

În funcție de implementarea aleasă pentru verificarea primalității se pot obține punctaje gradual, între $20 - 50p$ ($O(n)$, $O(sqrt(n))$, ciurul lui Eratostene)

Cerința 2 - 30p

Pe baza șirului A, construim șirul B folosind un algoritm de descompunere în factori primi, unde fiecare element al șirului B_i este cel mai mic număr natural care are aceiași factori primi cu A_i, cu $1 \leq i \leq n$.

Algoritmul de descompunere în factori primi va determina, pentru fiecare element al șirului A, atât elementele șirului B cât și numărul de factori primi al fiecărui element B_i. Se va reține cel mai mare număr al șirului B care are un număr maxim de factori primi.

În funcție de implementarea aleasă pentru descompunerea în factori primi se pot obține punctaje gradual, între $10 - 30p$.

Cerința 3 - 20p

Pentru determinarea lungimii maxime a unei secvențe mandatorii din șirul B se poate folosi o structură *for* ce parcurge numerele mandatare. Acestea pot fi: 2, 3, 5, 6 sau 7.

2.6 Cod-sursă pentru problema Mandatar

```
#include <bits/stdc++.h>
using namespace std;
ifstream  fcin("mandatar.in");
ofstream fcout("mandatar.out");
const int NM = 1e7;
int b[100001];
int c, n, x, Max, nrf, nrf_max, MaxM, k, kmax, ok;
int ma[] = {0,2,3,5,6,7};
void mandatar(int x, int &y, int &nf)
{ int d = 2;
  nf = 0; y = 1;
  if (x % d == 0)
     { y = 2; nf = 1;
       while (x % d == 0) x = x / d;
     }
   d = 3;
   while (d*d <= x)
         {
          if (x%d == 0)
             {nf++; y *= d;
              while (x % d == 0) x = x / d;
             }
          else d = d+2;
          }
    if (x > 1) { nf++; y *= x; }
}
int main()
{ fcin >> c >> n;
  for (int i=1; i<=n; i++)
       {fcin >> x;
        mandatar(x, b[i], nrf);
        if (nrf == 1) Max = max(Max, x);
        if (nrf > nrf_max) nrf_max = nrf, MaxM = b[i];
        else
           if (nrf == nrf_max && b[i] > MaxM) MaxM = b[i];
       }
  if (c == 1) fcout << Max;
  else
      if (c == 2) fcout << MaxM;
      else
         {for (int j=1; j<=5; j++)
              {ok = k = 0;
               for (int i=1; i<=n; i++)
                   {
                    if (b[i] == ma[j]) ok = 1;
                    if (b[i] % ma[j] == 0) k++;
                    else
                       k = 0, ok = 0;
                    if (k > kmax && ok) kmax = k;
                   }
              }
           fcout << kmax;
          }
    return 0;
}
```

Capitolul 3

OJI 2025, clasa a VII-a

3.1 Problema Prietenie

Propusă de: stud. Marcu Mihai, Delft University of Technology

Elevii celor două clase de a șaptea din școală merg în excursie. În fiecare clasă sunt câte N elevi. Ovidiu și Mihnea, fiind liderii celor două clase din care fac parte, doresc să analizeze reușita excursiei, în funcție de gradul de compatibilitate dintre elevii participanți la excursie.

Pentru a determina acest grad, fiecărui elev din cele două clase îi este atribuit un coeficient de amabilitate. Astfel, elevii din clasa lui Ovidiu au, în ordinea din catalog, coeficienții $a_1, a_2, \ldots, a_N$, iar elevii din clasa lui Mihnea au, în ordinea din catalog, coeficienții $b_1, b_2, \ldots, b_N$.

Gradul de compatibilitate dintre doi elevi din clase diferite este definit ca pătratul diferenței dintre coeficienții de amabilitate atribuiți fiecăruia. Astfel, gradul de compatibilitate G_{ij} dintre al i-lea elev din clasa lui Ovidiu și al j-lea elev din clasa lui Mihnea este egal cu $(a_i - b_j)^2$, cu $1 \leq i \leq N$ și $1 \leq j \leq N$.

Gradul de compatibilitate dintre cele două clase este suma tuturor gradelor de compatibilitate dintre oricare doi elevi din clase diferite, adică suma tuturor valorilor G_{ij} cu $1 \leq i \leq N$ și $1 \leq j \leq N$.

Pentru a lega o **prietenie durabilă** doi elevi din clase diferite trebuie să aibă gradul de compatibilitate fie mai mic sau egal cu X, fie mai mare sau egal cu Y, unde X și Y sunt valori date (adică $G_{ij} \leq X$ sau $Y \leq G_{ij}$)

Se cunosc N, coeficienții $a_1, a_2, \ldots, a_N$ și $b_1, b_2, \ldots, b_N$, precum și valorile X și Y, cu semnificația din enunț.

Cerințe

1. Determinați gradul de compatibilitate dintre cele două clase.
2. Determinați, pentru fiecare elev din clasa lui Ovidiu, numărul de elevi din clasa lui Mihnea cu care acesta poate lega o prietenie durabilă.

Date de intrare

Fișierul `prietenie.in` conține pe prima linie un singur număr natural C, semnificând cerința care trebuie rezolvată (care poate fi doar 1 sau 2). Pe a doua linie se găsesc trei numere naturale N, X și Y, cu semnificația din enunț. Pe a treia linie se găsesc N numere naturale $a_1, a_2, \ldots, a_N$, cu

semnificația din enunț. Pe a patra linie se găsesc N numere naturale $b_1, b_2, \ldots, b_N$, cu semnificația din enunț. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Fișierul `prietenie.out` conține:

- dacă $C = 1$, numărul natural determinat pentru cerința 1;
- dacă $C = 2$, N numere naturale, separate prin câte un spațiu, reprezentând numerele determinate pentru cerința 2, corespunzătoare ordinii în care elevii apar în catalogul clasei.

Restricții

- $1 \leq N \leq 200\,000$.
- $0 \leq a_i, b_j \leq 30\,000$, pentru oricare $1 \leq i, j \leq N$.
- $0 \leq X \leq Y \leq 900\,000\,000$.
- Ovidiu a observat că $(a_i - b_j)^2$ se poate scrie și sub forma $a_i^2 + b_j^2 - 2 \cdot a_i \cdot b_j$.

#	Puncte	Restricții
1	25	$C = 1$ și $1 \leq N \leq 2\,500$
2	10	$C = 1$, $2\,500 < N \leq 200\,000$ și $0 \leq a_i, b_j \leq 5\,000$, pentru oricare $1 \leq i, j \leq N$
3	10	$C = 1$ și $2\,500 < N \leq 200\,000$
4	25	$C = 2$ și $1 \leq N \leq 2\,500$
5	15	$C = 2$, $2\,500 < N \leq 200\,000$ și $0 \leq a_i, b_j \leq 5\,000$, pentru oricare $1 \leq i, j \leq N$
6	15	$C = 2$ și $2\,500 < N \leq 200\,000$

Exemple

prietenie.in	prietenie.out	Explicații
1 4 3 10 1 3 5 7 5 1 4 2	136	Se rezolvă cerința $C = 1$. Avem $N = 4$. $G_{11} = (a_1 - b_1)^2 = (1-5)^2 = 16$; $G_{12} = (a_1 - b_2)^2 = (1-1)^2 = 0$; ... Gradul de compatibilitate dintre cele două clase este egal cu: $(1-5)^2 + (1-1)^2 + (1-4)^2 + (1-2)^2 +$ $(3-5)^2 + (3-1)^2 + (3-4)^2 + (3-2)^2 +$ $(5-5)^2 + (5-1)^2 + (5-4)^2 + (5-2)^2 +$ $(7-5)^2 + (7-1)^2 + (7-4)^2 + (7-2)^2 =$ $16 + 0 + 9 + 1 + 4 + 4 + 1 + 1 + 0 + 16 +$ $1 + 9 + 4 + 36 + 9 + 25 = 136$

2 4 3 10 1 3 5 7 5 1 4 2	3 2 3 2	Se rezolvă cerința $C = 2$, pentru care $N = 4$, $X = 3$ și $Y = 10$. Pentru primul elev din clasa lui Ovidiu, care are coeficientul $a_1 = 1$, gradele de compatibilitate cu elevii din cealaltă clasă sunt: • $(a_1 - b_1)^2 = (1-5)^2 = 16$, iar $16 \geq Y$; • $(a_1 - b_2)^2 = (1-1)^2 = 0$, iar $0 \leq X$; • $(a_1 - b_3)^2 = (1-4)^2 = 9$, iar $X < 9 < Y$; • $(a_1 - b_4)^2 = (1-2)^2 = 1$, iar $1 \leq X$; Astfel, el poate lega o prietenie de lungă durata cu 3 elevi din clasa lui Mihnea: cu primul, cu al doilea și cu al patrulea. Al doilea elev din clasa lui Ovidiu poate lega o prietenie de lungă durata cu doi elevi din clasa lui Mihnea: cu al treilea și cu al patrulea. Analog, al treilea și al patrulea elev din clasa lui Ovidiu pot lega o prietenie de lungă durată cu 3 elevi, respectiv cu 2 elevi din clasa lui Mihnea.

3.2 Rezolvarea problemei Prietenie

Cerința 1

O observație necesară pentru rezolvarea primei cerințe este că $(a_i - b_j)^2$ se poate scrie sub forma $a_i^2 - 2 \cdot a_i \cdot b_j + b_j^2$, iar o altă observație e că fiecare număr din șirul a, va fi implicat în n sume (cu fiecare element din șirul b). Deci pentru fiecare element a_i, vom avea suma:

$$(a_i - b_1)^2 + (a_i - b_2)^2 + ... + (a_i - b_n)^2 =$$

$$= (a_i^2 - 2 \cdot a_i \cdot b_1 + b_1^2) + (a_i^2 - 2 \cdot a_i \cdot b_2 + b_2^2) + ... + (a_i^2 - 2 \cdot a_i \cdot b_n + b_n^2) =$$
$$= n \cdot a_i^2 - 2 \cdot a_i \cdot (b_1 + b_2 + ... + b_n) + (b_1^2 + b_2^2 + ... + b_n^2) =$$
$$= n \cdot a_i^2 - 2 \cdot a_i \cdot SumB + SumPatrateB$$

unde $SumB = b_1 + b_2 + ... + b_n$ este constant și se va calcula separat. Similar, $SumPatrateB = b_1^2 + b_2^2 + ... + b_n^2$. Astfel se vor calcula pentru fiecare termen al șirului a aceste sume și se vor adăuga la suma totală. Complexitatea finală este $O(n)$.

Cerința 2

Pentru fiecare element al șirului a (a_i), vom căuta câte elemente din șirul b (b_j), au proprietatea că $(a_i - b_j)^2$ este fie mai mic sau egal cu X, fie mai mare sau egal cu Y. Pentru a obține acest lucru se vor calcula frecvențele din șirul b și sumele parțiale pe ele. Așadar, considerăm fr_i frecvența fiecărui număr din șirul b, iar $sumfr_i = fr_1 + fr_2 + ... + fr_i$. Vom calcula câte numere avem astfel încât $(a_i - b_j)^2 <= X$, $-\sqrt{X} <= b_j - a_i <= \sqrt{X}$, adică $a_i - \sqrt{X} <= b_j <= a_i + \sqrt{X}$. Pentru a calcula numărul de numere dintre aceste 2 valori se va folosi șirul de sume parțiale $sumfr_i$. Acum vom calcula câte numere din șirul b există astfel încât $(a_i - b_j)^2 >= Y$, adică $b_j - a_i >= \sqrt{Y}$

sau $b_j - a_i <= -\sqrt{Y}$, deci $b_j >= \sqrt{Y} + a_i$ sau $b_j <= -\sqrt{Y} + a_i$, cu ajutorul șirului $sumfr_i$. Complexitatea finală este $O(n)$.

3.3 Cod-sursă pentru problema Prietenie

```
/* Mihai Marcu, Student TU Delft*/
#include <bits/stdc++.h>
using namespace std;

ifstream f("prietenie.in");
ofstream g ("prietenie.out");

int n,c;
int p,q;
int a[200005];
int b[200005];

long long sumA;
long long rez1;

int rez[200005];
int frecvB[300005];
int sumpartFrecv[300005];

int main()
{
    f>>c;
    f>>n;
    f>>p>>q;
    for(int i=1;i<=n;++i)
        f>>a[i];
    for(int i=1;i<=n;++i)
        f>>b[i];

    if(c==1){
        for(int i=1;i<=n;++i){
            rez1+=1LL*n*a[i]*a[i];
            rez1+=1LL*n*b[i]*b[i];
            sumA+=a[i];
        }
        for(int i=1;i<=n;++i){
            rez1-=1LL*2*sumA*b[i];
        }
        g<<rez1;
    }
    else{
        for(int i=1;i<=n;++i)
            frecvB[b[i]]++;

        sumpartFrecv[0]=frecvB[0];
        for(int i=1;i<=300000;++i)
        {
            sumpartFrecv[i]=sumpartFrecv[i-1]+frecvB[i];
        }

        int stQ,drQ,stP,drP;
        for(int i=1;i<=n;++i){
            stQ=-1; drQ=-1;
            stP=-1; drP=-1;
```

```
            stP=a[i]-min(a[i], (int)sqrt(p));
            stQ=a[i]-min(a[i]+1, (int)sqrt(q-1)+1);
            drP=(int)sqrt(p)+a[i];
            drQ=(int)sqrt(q-1)+1+a[i];
            int x=0;
            if(stQ>=0){
               x+=sumpartFrecv[stQ];
            }
            if(stP>=0)
               x+=(sumpartFrecv[a[i]-1]-sumpartFrecv[stP-1]);

            x+=(sumpartFrecv[drP]-sumpartFrecv[a[i]-1]);
            x+=sumpartFrecv[30000]-sumpartFrecv[drQ-1];
            g<<x<<" ";
        }

    }
    return 0;
}
```

```
// prof. Panaete Adrian, Colegiul Național "August Treboniu Laurian", Botoșani
#include <bits/stdc++.h>
using namespace std;
ifstream f("prietenie.in");
ofstream g("prietenie.out");
const int N = 200010;
const int M = 30002;
int cer,n,x,X,Y,A,B,C,D,O[N],S[M],P[M];
int64_t r,a,b;
int sum(int a,int b){
  if(b<=0)return 0;
  if(a>=M)return 0;
  if(a<=0)a=1;
  if(b>=M)b=M-1;
  if(a>b)return 0;
  return S[b]-S[a-1];
}
int main()
{
  f>>cer>>n>>X>>Y;
  for(int i=1;i<=n;i++){f>>x;r+=x*x;a+=x;O[i]=++x;}
  for(int i=1;i<=n;i++){f>>x;r+=x*x;b+=x;S[++x]++;}
  if(cer==1){r=r*n-a*b*2;g<<r;return 0;}
  for(int i=1;i<M;i++)S[i]+=S[i-1];
  for(C=0;C*C<=X;C++,B--);for(A=B,D=C;D*D<Y;D++,A--);A++;D--;
  for(int i=1;i<M;i++){A++;B++;C++;D++;P[i]=n-sum(A,B)-sum(C,D);}
  for(int i=1;i<=n;i++)g<<P[O[i]]<<' ';g<<'\n';
  return 0;
}
```

3.4 Problema Teren

Propusă de: prof. Popa Daniel, Colegiul Național „Aurel Vlaicu", Orăștie

Lordul John a decis că a venit vremea să însămânțeze terenul său. Terenul a fost împărțit în parcele organizate în N linii, pe fiecare linie fiind câte N parcele pătrate, fiecare cu suprafața de un metru pătrat. Liniile au fost numerotate de sus în jos de la 1 la N, iar coloanele de la stânga la dreapta de la 1 la N.

Fiind un aviator pasionat, a folosit avionul său pentru a survola terenul în vederea însămânțării. Când se află în zbor deasupra câte unei parcele, aruncă în aceasta o singură sămânță. Lordul realizează M zboruri deasupra terenului, iar în fiecare astfel de zbor se deplasează în câte o singură direcție, paralelă cu laturile sau cu diagonalele terenului. Fiecare zbor i (cu $1 \leq i \leq M$) este definit printr-un set de patru valori, LS_i CS_i LF_i CF_i, unde (LS_i, CS_i) sunt coordonatele (linia și coloana) primei parcele în care a aruncat o sămânță și (LF_i, CF_i) sunt coordonatele parcelei în care a aruncat ultima sămânță, în cadrul acestui zbor.

La final, după însămânțare, Lordul John dorește să împrejmuiască cu gard parcelele însămânțate, pentru a le separa de cele rămase neînsămânțate sau de marginea terenului.

Se cunosc N, M, precum și valorile (LS_1, CS_1, LF_1, CF_1), (LS_2, CS_2, LF_2, CF_2), ..., (LS_M, CS_M, LF_M, CF_M) cu semnificația din enunț.

Cerințe

1. Determinați numărul semințelor care sunt aruncate.
2. Determinați numărul de parcele care sunt însămânțate.
3. Determinați lungimea gardului care trebuie să separe suprafețele însămânțate de cele neînsămânțate sau de marginea terenului.

Date de intrare

Fișierul `teren.in` conține pe prima linie trei numere naturale, C, N și M, unde C este numărul cerinței care trebuie rezolvată (care poate fi doar 1, 2 sau 3), iar N și M au semnificația din enunț.

Pe următoarele M linii se află câte patru numere naturale, reprezentând seturile de valori care definesc zborurile, în ordinea realizării lor.

Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

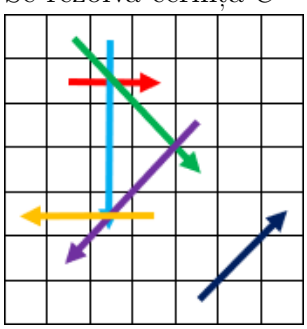
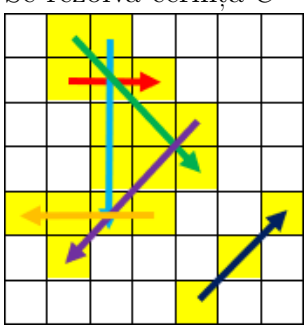
Fișierul `teren.out` conține numărul determinat pentru cerința C.

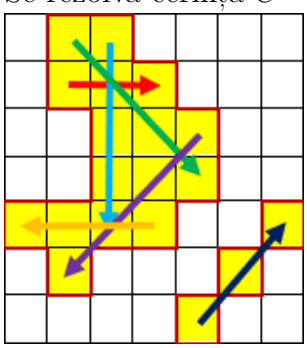
Restricții

- $1 \leq N, LS, LF, CS, CF \leq 1\,000$.
- $1 \leq M \leq 100\,000$

#	Puncte	Restricții
1	20	$C = 1$, avionul se deplasează numai de la stânga la dreapta sau de sus în jos
2	15	$C = 1$, avionul se poate deplasa în orice direcție
3	20	$C = 2$, avionul se deplasează numai de la stânga la dreapta sau de sus în jos
4	15	$C = 2$, avionul se poate deplasa în orice direcție
5	30	$C = 3$

Exemple

teren.in	teren.out	Explicații
1 7 6 2 2 2 4 1 3 5 3 1 2 4 5 3 5 6 2 5 4 5 1 7 5 5 7	23	Se rezolvă cerința $C = 1$.  Primul zbor este marcat de săgeata roșie; se survolează parcelele de la coordonatele (2, 2), (2, 3) și (2, 4); la acest zbor se aruncă 3 semințe. Al doilea zbor este marcat de săgeata albastru-deschis; la acest zbor se aruncă 5 semințe. În total, în cele 6 zboruri sunt aruncate 3 + 5 + 4 + 4 + 4 + 3 = 23 de semințe.
2 7 6 2 2 2 4 1 3 5 3 1 2 4 5 3 5 6 2 5 4 5 1 7 5 5 7	19	Se rezolvă cerința $C = 2$.  Parcelele însămânțate sunt colorate cu galben.

3 7 6 2 2 2 4 1 3 5 3 1 2 4 5 3 5 6 2 5 4 5 1 7 5 5 7	36	Se rezolvă cerința $C = 3$.  Gardurile folosite sunt marcate cu culoarea roșie, aflate la marginea unor parcele colorate cu galben.

3.5 Rezolvarea problemei Teren

Cerința 1

Se citesc coordonatele de început ale însămânțării $L1$, $C1$ și coordonatele de sfârșit $L2$, $C2$. Dacă $L1$==$L2$ la numărul de semințe aruncate se adună abs($C1$-$C2$)+1, altfel se adună abs($L1$ - $L2$) + 1.

Precalcule pentru Cerințele 2 și 3

Atât pentru cerința 2 cât și pentru cerința 3 se folosesc 5 matrici: matricea o pentru parcurgerile orizontale, v pentru cele verticale, dp pentru cele paralele cu diagonala principală, ds pentru parcurgerile paralele cu diagonala secundară. Pentru fiecare zbor/parcurgere se aplică difference array. Deoarece parcurgerea este liniară difference array se aplică exact ca la vectori: Pentru parcurgerile orizontale (unde $L1 == L2$):

$$\begin{cases} o[L1][min(C1, C2)] + +; \\ o[L1][max(C1, C2) + 1] - -; \end{cases}$$

Pentru parcurgerile verticale (unde $C1 == C2$):

$$\begin{cases} v[min(L1, L2)][C1] + +; \\ v[max(L1, L2) + 1][C1] - -; \end{cases}$$

Pentru parcurgerile paralele cu diagonala secundară (unde $L1+C1 == L2+C2$): se interschimbă capetele a.î. $L1 < L2$ și apoi:

$$\begin{cases} ds[L1][C1] + +; \\ ds[L2 + 1][C2 - 1] - -; \end{cases}$$

Pentru parcurgerile paralele cu diagonala principală: se interschimbă capetele a.î. $L1 < L2$ și apoi

$$\begin{cases} dp[L1][C1]++; \\ dp[L2+1][C2+1]--; \end{cases}$$

Se parcurg matricele și se fac adunările corespunzătoare:

$$\begin{cases} o[i][j] += o[i][j-1]; \\ v[i][j] += v[i-1][j]; \\ dp[i][j] += dp[i-1][j-1]; \\ ds[i][j] += ds[i-1][j+1]; \end{cases}$$

Într-o matrice rezultat se marchează acele celule care sunt nenule în cel puțin una din matricele anterioare.

Cerința 2

Pentru obținerea rezultatului se numără câte valori nenule sunt în matricea rezultat.

Cerința 3

Pentru a obține rezultatul se numără pentru fiecare celulă nenulă câți vecini au valoarea 0.

3.6 Cod-sursă pentru problema Teren

```
// prof Popa Daniel, Colegiul Național "Aurel Vlaicu", Orăștie
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;
ifstream fin("teren.in");
ofstream fout("teren.out");
const int nm=1002;
int o[nm][nm], v[nm][nm], dp[nm][nm], ds[nm][nm], r[nm][nm], n, m, k, i, j, L1, C1, L2, C2, c;
void afis(int a[nm][nm])
{
  for(int i=1; i<=n; i++)
    {
      for(j=1; j<=n; j++) cout << a[i][j] << " ";
      cout << endl;
    }
  cout << endl;
}
void cerinta1()
{
  int sol=0;
  for(int i=1; i<=m; i++)
    {
      fin >> L1 >> C1 >> L2 >> C2;
      if(L1==L2)sol+=abs(C1-C2)+1;
      else sol+=abs(L1-L2)+1;
    }
  fout << sol;
}
void pentruCerinta2_3()
```

```
{
  for(int i=1; i<=m; i++)
    {
      fin >> L1 >> C1 >> L2 >> C2;
      if(L1==L2){o[L1][min(C1, C2)]++; o[L1][max(C1, C2)+1]--;}
      else if(C1==C2){v[min(L1, L2)][C1]++; v[max(L1, L2)+1][C1]--;}
      else if(L1+C1==L2+C2) /// diagonala secundara
        {
          if(L1>L2){swap(L1, L2); swap(C1, C2);}
          ds[L1][C1]++; ds[L2+1][C2-1]--;
        }
      else /// diagonala principala
        {
          if(L1>L2){swap(L1, L2); swap(C1, C2);}
          dp[L1][C1]++; dp[L2+1][C2+1]--;
        }
    }
  for(i=1; i<=n; i++)
    for(j=1; j<=n; j++)
      {
        o[i][j]+=o[i][j-1];
        v[i][j]+=v[i-1][j];
        dp[i][j]+=dp[i-1][j-1];
        ds[i][j]+=ds[i-1][j+1];
        r[i][j]=(o[i][j] + v[i][j] + dp[i][j] + ds[i][j])>0;
      }
}

void cerinta2()
{   int sol=0;
  for(int i=1; i<=n; i++)
    for(j=1; j<=n; j++)
      if(r[i][j]>0)sol++;
  fout << sol;
}

void cerinta3()
{   int sol=0, k;
  fin >> k;
  for(int i=1; i<=n; i++)
    for(j=1; j<=n; j++)
      if(r[i][j]!=0)
        {
          sol+=(r[i-1][j]==0)+(r[i][j-1]==0)+(r[i+1][j]==0)+(r[i][j+1]==0);
        }
  fout << sol;
}

int main()
{
  fin >> c >> n >> m;
  if(c==1)cerinta1();
  else
    {
      pentruCerinta2_3();
      if(c==2)cerinta2();
      if(c==3)cerinta3();
    }
  return 0;
}

/* prof. Raluca Costineanu, Colegiul National Stefan cel Mare, Suceava*/
```

```cpp
#include <fstream>

using namespace std;

ifstream f("teren.in");
ofstream g("teren.out");
#define N 1005
int n, m, C;
int lin[N][N], col[N][N], diagP[N][N], diagS[N][N], tot[N][N];
int main()
{
  f >> C >> n >> m;
  int i, j, xs, ys, xd, yd;
  if(C == 1)
    {
      int total = 0;
      for(i = 1; i <= m; ++i)
        {
          f >> xs >> ys >> xd >> yd;
          if(xs == xd) total += abs(ys - yd) + 1;
          else if(ys == yd) total += abs(xs - xd) + 1;
          else total += abs(xs - xd) + 1;
        }
      g << total << '\n';
    }
  else
    {
      int total = 0;
      for(i = 1; i <= m; ++i)
        {
          f >> xs >> ys >> xd >> yd;
          if(xs == xd)
            {
              if(ys > yd) swap(ys, yd);
              lin[xs][ys]++;
              lin[xs][yd + 1]--;
            }
          else if(ys == yd)
            {
              if(xs > xd) swap(xs, xd);
              col[xs][ys]++;
              col[xd + 1][ys]--;
            }
          else
            {
              if(xs > xd) swap(xs,xd), swap(ys,yd);
              if(yd >= ys) diagP[xs][ys]++, diagP[xd + 1][yd + 1]--;
              else diagS[xs][ys]++, diagS[xd + 1][yd - 1]--;
            }
        }
      for(i = 1; i <= n; ++i)
        for(j = 1; j <= n; ++j)
          {
            lin[i][j] += lin[i][j - 1];
            col[i][j] += col[i - 1][j];
            diagP[i][j] += diagP[i - 1][j - 1];
            diagS[i][j] += diagS[i - 1][j + 1];
            if(lin[i][j] > 0 || col[i][j] > 0 || diagP[i][j] > 0 || diagS[i][j] > 0)
              tot[i][j] = 1;
            if(tot[i][j]) ++total;
          }
```

```
      if(C == 2)
        g << total << '\n';
      else
        {
          total = 0;
          for(i = 1; i <= n; ++i)
            for(j = 1; j <= n; ++j)
              if(tot[i][j])
                {
                  if(!tot[i - 1][j])total++;
                  if(!tot[i + 1][j])total++;
                  if(!tot[i][j - 1])total++;
                  if(!tot[i][j + 1])total++;
                }
          g << total << '\n';
        }
    }
  return 0;
}
```

Capitolul 4

OJI 2025, clasa a VIII-a

4.1 Problema Joc

Propusă de: stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București

Jocul preferat al lui Aurel are o hartă împărțită în N sectoare, numerotate, în ordine, de la 1 la N. Fiecare sector i ($1 \leq i \leq N$) are asociate două numere naturale reprezentând un decor, $decor_i$ și un scor, $scor_i$. Două decoruri de același tip sunt codificate prin același număr natural.

O secvență formată din lg ($lg \geq 2$) sectoare aflate pe poziții consecutive este numită **riscantă** dacă cel puțin $\frac{lg}{2} + 1$ dintre sectoarele acesteia au asociat același tip de decor, unde $\frac{lg}{2}$ reprezintă câtul împărțirii lui lg la 2.

Dacă Aurel se află pe sectorul s și are vizibilitatea v ($0 \leq v \leq s - 1$), el va „vedea" pe hartă secvența de $v + 1$ sectoare consecutive, care se încheie cu s: $s - v, s - v + 1, \ldots s$.

La începutul jocului, Aurel este poziționat într-un anumit sector (sector de start) și are o anumită vizibilitate. La fiecare pas al jocului, Aurel, fiind poziționat într-un sector oarecare, efectuează una dintre acțiunile:

- dacă secvența pe care o „vede" pe hartă este riscantă, Aurel scade cu 1 vizibilitatea pe care o are (astfel el speră ca secvența rezultată să nu mai fie riscantă);
- dacă secvența pe care o „vede" pe hartă nu este riscantă, Aurel avansează, poziționându-se în sectorul următor, și crește cu 1 vizibilitatea (el se simte încurajat și merge mai departe).

Jocul se termină când el iese de pe hartă, adică se află după sectorul cu numărul N (ultimul).

Scorul obținut este egal cu suma scorurilor sectoarelor în care el a fost poziționat la fiecare pas pe parcursul jocului (inclusiv scorul sectorului de start).

Cerințe

1. Determinați numărul de moduri în care Aurel poate începe jocul, astfel încât prima secvență pe care o „vede" pe hartă să NU fie riscantă. Două moduri de a începe jocul sunt considerate diferite dacă încep pe sectoare diferite sau dacă au vizibilitatea diferită.
2. Determinați scorul obținut dacă Aurel pornește din sectorul 1 cu vizibilitatea 0.

Date de intrare

Fișierul de intrare `joc.in` conține pe prima linie numărul natural C reprezentând cerința care trebuie să fie rezolvată (1 sau 2). Pe a doua linie se află numărul natural N reprezentând numărul

de sectoare. Pe a treia linie se află N numere naturale, reprezentând decorurile asociate sectoarelor, în ordinea numerotării acestora. Pe a patra linie se află N numere naturale, reprezentând scorurile asociate sectoarelor, în ordinea numerotării acestora. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `joc.out` conține o singură linie pe care este scris numărul determinat pentru cerința C din fișierul de intrare.

Restricții

- Dacă $C = 1$, atunci $1 \leq N \leq 3\,000$
- Dacă $C = 2$, atunci $1 \leq N \leq 100\,000$
- $1 \leq decor_i \leq N$, pentru $1 \leq i \leq N$
- $1 \leq scor_i \leq 1\,000\,000$, pentru $1 \leq i \leq N$

#	Puncte	Restricții	
1	25	$C = 1$,	$1 \leq N \leq 800$
2	21	$C = 1$,	$800 < N \leq 3\,000$
3	24	$C = 2$,	$1 \leq N \leq 9\,000$
4	30	$C = 2$,	$9000 < N \leq 100\,000$

Exemple

joc.in	joc.out
1 5 1 1 2 1 3 2 3 1 1 5	10
2 5 1 1 2 1 3 2 3 1 1 5	16

Explicație

Exemplul 1. Se notează cu st sectorul de start și cu v vizibilitatea; există 10 moduri în care Aurel poate începe jocul astfel încât prima secvență văzută să nu fie riscantă:

1. $st = 1$, $v = 0$ (secvența 1)
2. $st = 2$, $v = 0$ (secvența 2)
3. $st = 3$, $v = 0$ (secvența 3)
4. $st = 4$, $v = 0$ (secvența 4)
5. $st = 5$, $v = 0$ (secvența 5)
6. $st = 3$, $v = 1$ (secvența 2,3)
7. $st = 4$, $v = 1$ (secvența 3,4)
8. $st = 5$, $v = 1$ (secvența 4,5)
9. $st = 5$, $v = 2$ (secvența 3,4,5)
10. $st = 5$, $v = 3$ (secvența 2,3,4,5)

Exemplul 2. Aurel pornește din sectorul 1 cu vizibilitate $v = 0$. Scorul total este inițial $scor_1 = 2$.

- El vede doar sectorul 1, iar secvența văzută nu este riscantă, deci avansează în sectorul 2 și crește v cu 1. La scorul total se adună $scor_2 = 3$.
- Secvența văzută, formată din sectoarele 1,2, este riscantă, deci scade v cu 1. Aurel este acum în sectorul 2, cu $v = 0$. La scorul total se adună $scor_2 = 3$.
- Secvența curentă văzută nu este riscantă, deci avansează în sectorul 3 și crește v cu 1. La scorul total se adună $scor_3 = 1$.
- Secvența văzută 2,3 nu este riscantă, deci avansează în sectorul 4 și crește v cu 1. La scorul total se adună $scor_4 = 1$.
- Secvența văzută, formată din sectoarele 2,3,4 este riscantă, deci scade v cu 1. La scorul total se adună $scor_4 = 1$.
- Secvența văzută 3,4 nu este riscantă, deci avansează în sectorul 5 și crește v cu 1. La scorul total se adună $scor_5 = 5$.
- Secvența văzută formată din sectoarele 3,4,5 nu este riscantă, deci avansează și se poziționează după ultimul sector, terminând jocul. Scorul total obținut este $2 + 3 + 3 + 1 + 1 + 1 + 5 = 16$.

4.2 Rezolvarea problemei Joc

Cerința 1 – O(N^3)

Vom fixa cele două valori st ($1 \leq st \leq n$) și v ($0 \leq v < st$) reprezentând începutul secvenței și vizibilitatea. Pentru fiecare secvență determinată de st și v vom aplica unul dintre algoritmii liniari de calculare a elementului majoritar (resursă: Infoarena - Problema majorității votului). Astfel putem verifica, pentru fiecare secvență dacă este riscantă.

Cerința 1 – O(N^2)

Vom folosi un vector de frecvență, în care vom contoriza numărul de apariții pentru fiecare decor. Să considerăm că am determinat vectorul de frecvență pentru secvența $[i,j]$ care începe la poziția i și se termină la poziția j ($1 \leq i \leq j < n$). Când vom trece la secvența $[i,j+1]$ vom adăuga un singur element, deci putem actualiza ușor vectorul de frecvență. Elementul majoritar se poate recalcula în momentul în care adăugăm un element în vectorul de frecvență (fie rămâne valoarea precedentă, fie devine noua valoare adăugată). După ce toate secvențele cu capăt stâng i au fost analizate, resetăm vectorul de frecvență pentru a-l refolosi pentru subsecvențele cu capătul stâng $i + 1$.

Cerința 2 – O(N^2)

Vom simula efectiv jocul. Pentru a afla dacă o subsecvență este sau nu riscantă vom folosi un algoritm liniar de aflare a elementului majoritar (similar soluției de la cerința 1). Acestă soluție are complexitatea O(N^2) în cazul cel mai defavorabil.

Cerința 2 – O(N)

Pentru a optimiza determinarea elementului majoritar din soluția precedentă, vom folosi un vector de frecvență, similar soluției 2 de la cerința 1. Să presupunem că ajungem pe subsecvența $[s-v,s]$ și știm dacă aceasta conține sau nu element majoritar, care este acesta și numărul său de apariții. Avem două cazuri:

- Secvența este riscantă. În acest caz știm că există element majoritar (fie acesta $emax$). Jucătorul va micșora vizibilitatea, astfel excluzând elementul de pe poziția $s-v$. Datorită faptului că elementul $emax$ avea $(v+1)/2+1$ apariții în subsecvența $[s-v,s]$, acesta rămâne elementul cu număr maxim de apariții și în subsecvența $[s-v+1,s]$, deci ar fi posibil ca acesta să rămână element majoritar sau să nu mai existe element majoritar.
- Secvența nu este riscantă. În acest caz știm că nu există element majoritar. Când adăugăm un element în subsecvență avem două cazuri posibile. Fie acesta are acum numărul necesar de apariții, caz în care actualizăm $emax$, fie acesta nu are suficiente apariții, caz în care nu avem element majoritar.

Complexitatea acestei soluții este $O(N)$ timp și $O(N)$ memorie. Există și alte soluții, atât pentru punctaj integral, cât și pentru punctaje parțiale.

4.3 Cod-sursă pentru problema Joc

```
#include<cstdio>
const int NMAX=100005, NMAX2=1024;
int N;
int decor[NMAX], scor[NMAX], cnt[NMAX];
int cerinta_1()
{ int i, j, maxAp, rez=0;
  for (i=0; i<N; ++i)
      {
       for (j=i, maxAp=decor[i]; j<N; ++j)
           {
            if (++cnt[decor[j]]>cnt[maxAp]) maxAp=decor[j];
            if (!(i<j && cnt[maxAp]>(j-i+1)/2)) ++rez;
           }
       for (j=N-1;j>=i;--j) --cnt[decor[j]];
      }
  return rez;
}
int cerinta_2()
{ int i, j, maxAp, total=0;
  for (i=j=0, ++cnt[maxAp=decor[0]]; j<N; )
          { total+=scor[j];
        if (j>i && cnt[maxAp]>(j-i+1)/2) --cnt[decor[i++]];
                    else
           {++j;
            if (j<N)
               if (++cnt[decor[j]]>cnt[maxAp]) maxAp=decor[j];
           }
      }
  return total;
}
int main()
{ FILE* f=fopen("joc.in", "r"), *g=fopen("joc.out", "w");
  int i, C;
  fscanf(f, "%d%d", &C, &N);
  for (i=0;i<N;++i) fscanf(f, "%d", decor+i);
  for (i=0;i<N;++i) fscanf(f, "%d", scor+i);
  fprintf(g, "%d\n", C==1 ? cerinta_1() : cerinta_2());
  fclose(f); fclose(g);
  return 0;
}
```

4.4 Problema Reducere

Propusă de: prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași

O **operație de reducere** aplicată asupra unui șir constă în selectarea unui număr prim p și a unor elemente din șirul dat care sunt divizibile cu p și împărțirea acestora la p.

Asupra unui șir format din n numere naturale nenule se aplică o succesiune de operații de reducere, până când toate elementele șirului devin egale. Valoarea finală a elementelor șirului este denumită **valoare de egalitate**.

Valoarea de reducere a unui șir este cea mai mare dintre valorile de egalitate care se pot obține în urma aplicării unor operații de reducere asupra acestui șir.

Cerințe

1. Determinați valoarea de reducere pentru un șir dat.
2. Determinați numărul minim de operații de reducere care trebuie să fie aplicate șirului dat pentru a obține valoarea de reducere.

Date de intrare

Fișierul de intrare `reducere.in` conține pe prima linie un număr natural C, reprezentând cerința care trebuie să fie rezolvată (1 sau 2), pe a doua linie un număr natural n, iar pe următoarele n linii câte un număr natural, reprezentând câte un element al șirului.

Date de ieșire

Fișierul de ieșire `reducere.out` va conține o singură linie, pe care va fi scris numărul determinat pentru cerința C din fișierul de intrare.

Restricții

- $2 \leq n \leq 2000$
- Elementele șirului sunt numere naturale nenule $\leq 10^{12}$.

#	Puncte	Restricții
1	34	$C = 1$
2	37	$C = 2$, valorile din șir $\leq 10^6$
3	29	$C = 2$, fără restricții suplimentare

Exemple

reducere.in	reducere.out
1 3 10 6 12	2

2 3 10 6 12	3

Explicație

Operația de reducere 1: împărțim prin 3 pe 6 și pe 12 ⇒ 10 2 4
Operația de reducere 2: împărțim prin 5 pe 10 ⇒ 2 2 4
Operația de reducere 3: împărțim prin 2 pe 4 ⇒ 2 2 2
Valoarea de reducere este 2, aceasta fiind cea mai mare dintre valorile de egalitate posibile.

4.5 Rezolvarea problemei Reducere

Cerința 1.

Pentru a determina valoarea de reducere, trebuie să determinăm cel mai mare divizor comun al valorilor din secvență. Pentru a determina cmmdc pentru două valori utilizăm algoritmul lui Euclid. Pentru a determina cmmdc pentru o secvență de n valori utilizăm asociativitatea operației cmmdc, deci determinăm la fiecare pas cmmdc dintre cmmdc-ul curent și următoarea valoare din secvență: $cmmdc(a_1, a_2, \cdots a_n) = cmmdc(...(cmmdc(cmmdc(a_1, a_2), a_3) \cdots a_n) \cdots)$.

Cerința 2.

Descompunem în factori primi fiecare valoare din secvență și determinăm pe parcurs descompunerea în factori primi a celui mai mic multiplu comun al acestor valori (toți factorii primi care apar în descompunerile valorilor din secvență la puterea cea mai mare). Factorii primi comuni la puterea cea mai mică constituie cmmdc (deci îi păstrăm în valoarea de reducere). Numărul minim de operații care trebuie să fie aplicate pentru a obține valoarea de reducere este egal cu suma exponenților factorilor primi din descompunerea în factori primi a cmmmc/cmmdc.

Pentru subtask-ul 2, restricțiile permit utilizarea unui vector nr de 10^6 elemente, unde nr_i = puterea factorului prim i în descompunerea factori primi a cmmmc/cmmdc. Pentru a obține punctele pe acest subtask nu este necesar să optimizăm descompunerea în factori primi utilizând generarea prealabilă a numerelor prime cu ciurul lui Eratostene, dar, pentru subtask-ul 3, este necesar să descompunem în factori primi căutând divizorii, doar printre numerele prime până la radicalul numărului.

Pentru subtask-ul 3 restricțiile nu permit declararea vectorului nr. Ca urmare vom reține o descompunere în factori primi ca o listă de factori primi și puterile acestora, listă în care factorii primi apar în ordine crescătoare.

Pentru fiecare număr din secvență:

- descompunem numărul în factori primi;
- printr-un algoritm similar cu algoritmul de interclasare, actualizăm descompunerea în factori primi a cmmmc (în cmmmc trebuie să apară toți factorii primi la puterea cea mai mare).

La final simplificăm cmmmc cu cmmdc, parcurgând descompunerile în factori primi ale acestora și, pentru factorii primi comuni, scăzând din puterea factorului prim din cmmmc puterea factorului prim respectiv din cmmdc.

Suma puterilor factorilor primi ai cmmmc după simplificarea cu cmmdc va fi numărul minim de operații de reducere necesare.

4.6 Cod-sursă pentru problema Reducere

```
#include <fstream>
#define VMAX 1000000
#define PMAX 80000
#define LGMAX 8
#define NMAX 2002
using namespace std;
ifstream fin("reducere.in");
ofstream fout("reducere.out");
struct factor {long long int d; short int p;};
bool ciur[VMAX];
int prime[PMAX];
void eratostene();
long long int cmmdc ();
int c, n, m, nrp;
long long int cmd;
factor D[LGMAX], dx[LGMAX], rez[LGMAX*NMAX], aux[LGMAX*NMAX];
int lgD, lgx, lgrez;
long long int nr[NMAX];


long long int cerinta2();
void descompunere(long long int x, factor dx[], int & lgx);

int main()
{int i;
 fin>>c>>n;
 for (i=0; i<n; i++) fin>>nr[i];
 cmd=cmmdc();
 if (c==1)
    fout<<cmd<<'\n';
    else
    {eratostene();
     fout<<cerinta2()<<'\n';
    }
 return 0;
}

void eratostene()
{int i, j;
 for (i=3; i*i<VMAX; i+=2)
     if (ciur[i]==0)
         for (j=i*i; j<VMAX; j+=i)
             ciur[j]=1;
 prime[0]=2; nrp=1;
 for (i=3; i<VMAX; i+=2)
     if (ciur[i]==0) prime[nrp++]=i;
}

long long int cmmdc()
{int i;
 long long int d, x, r;
 d=nr[0];
 for (i=1; i<n; i++)
     {x=nr[i];
      while (x)
```

```
            {r=d%x;
             d=x;
             x=r;
            }
     }
 return d;
}

void descompunere(long long int x, factor dx[], int & lgx)
{int i, m;
 lgx=0;
 for (i=0; i<nrp && (long long int)prime[i]*prime[i]<=x; i++)
     if (x%prime[i]==0)
        {
         m=0; while (x%prime[i]==0) {m++; x/=prime[i];}
         dx[lgx].p=m; dx[lgx].d=prime[i]; lgx++;
        }
 if (x>1) {dx[lgx].d=x; dx[lgx].p=1; lgx++;}
}

void actualizeaza()
{int i=0, j=0, k=0;
 while (i<lgx && j<lgrez)
        if (dx[i].d==rez[j].d)
           {aux[k].d=dx[i].d; aux[k].p=max(dx[i].p, rez[j].p); i++; j++; k++;}
           else
           if (dx[i].d<rez[j].d)
              aux[k++]=dx[i++];
              else
              aux[k++]=rez[j++];
 while (i<lgx) aux[k++]=dx[i++];
 while (j<lgrez) aux[k++]=rez[j++];
 for (i=0; i<k; i++) rez[i]=aux[i];
 lgrez=k;
}
long long int cerinta2()
{int i, j;
 long long int nrop=0;
 descompunere(cmd, D,lgD);
 descompunere(nr[0],rez,lgrez);
 for (i=1; i<n; i++)
     {descompunere(nr[i],dx,lgx);
      actualizeaza();}
 i=0; j=0;
 while (i<lgD)
 if (D[i].d==rez[j].d) {rez[j].p-=D[i].p; i++; j++;}
     else j++;
 for (i=0; i<lgrez; i++)
     nrop+=rez[i].p;
 return nrop;
}
```

Partea a II-a

Olimpiada Națională de Informatică - etapa națională -

Botoșani, 14-18 aprilie 2025

Capitolul 5

ONI 2025, clasa a V-a

5.1 Problema Cartonașe

Propusă de: prof. Marius Nicoli, Colegiul Național "Frații Buzești", Syncro Soft, Craiova

Maria inventează mereu câte ceva și îl provoacă la joacă pe fratele ei mai mic Petru. De data aceasta alege N cartonașe, pe care sunt înscrise valorile naturale distincte de la 1 la N (fiecare astfel de număr apare pe câte un singur cartonaș), le amestecă și le așează unul lângă altul într-un șir. După amestecare numerotează cartonașele cu valori de la 1 la N, după ordinea așezării în șir. Apoi îi formulează diverse cerințe lui Petru. Petru a învățat să programeze și acum dorește să scrie un program pentru a-i răspunde Mariei repede și fără să se mai gândească mult.

Cerințe

Maria formulează lui Petru cerințe de următoarele tipuri:

1. Îți spun un număr *poz* și trebuie să determini cartonașul numerotat cu cea mai mare valoare r astfel încât primele r cartonașe din șir au înscrisă o valoare strict mai mică decât cea scrisă pe cartonașul numerotat cu *poz*. Dacă nu există niciun astfel de cartonaș, pentru r se stabilește valoarea 0.
2. Determină toate valorile p cu proprietatea că pe primele p cartonașe se află înscrise toate numerele naturale de la 1 la p.
3. Determină toate valorile p cu proprietatea că pe primele p cartonașe se află înscrise exact $p-1$ dintre numerele naturale de la 1 la p.

Date de intrare

Fișierul de intrare `cartonase.in` conține:

- pe prima linie numărul C, reprezentând cerința de rezolvat (1, 2 sau 3);
- pe linia a doua se află numărul N, cu semnificația din enunț;
- pe linia a treia, se află, separate prin câte un spațiu, N valori naturale distincte, cuprinse între 1 și N, reprezentând valorile înscrise pe cartonașe în ordinea din șir, după amestecare;
- dacă $C = 1$, pe linia a patra se află valoarea *poz*.

Numerele aflate pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `cartonase.out` conține:

- Dacă $C = 1$, în fișierul de ieșire se va afla valoarea r cu semnificația din enunț.
- Dacă $C = 2$ sau $C = 3$, în fișierul de ieșire se vor afișa, separate prin câte un spațiu, valorile lui p care îndeplinesc condițiile din cerința corespunzătoare, în ordine crescătoare. Se garantează că există cel puțin o astfel de valoare.

Restricții

- $1 \leq C \leq 3$;
- $1 \leq N \leq 100000$;
- $1 \leq poz \leq N$.

#	Puncte	Restricții
1	23	$C = 1$
2	41	$C = 2$
3	36	$C = 3$

Exemple

cartonase.in	cartonase.out	Explicații
1 6 3 1 6 2 4 5 5	2	$C = 1$, $poz = 5$, pe cartonașul 5 se află valoarea **4**. Primele două cartonașe din șirul dat au înscrise valori mai mici decât 4 iar al treilea are o valoare mai mare.
2 6 3 1 2 6 4 5	3 6	$C = 2$, pe primele 3 cartonașe se află valorile 1, 2, 3 și, de asemenea, pe primele 6 cartonașe se află valorile 1, 2, 3, 4, 5, 6.
3 6 3 1 2 6 5 4	1 2 4 5	$C = 3$, pe primul cartonaș ($p = 1$) se află $p - 1 = 0$ valori conform cerinței. Pe primele $p = 2$ cartonașe se află $p - 1 = 1$ valori conform cerinței (1). Pe primele $p = 3$ cartonașe se află 3 valori conform cerinței (1, 2, 3). Pe primele $p = 4$ cartonașe se află $p - 1 = 3$ valori conform cerinței (1, 2, 3). Pe primele $p = 5$ cartonașe se află $p - 1 = 4$ valori conform cerinței (1, 2, 3, 5). Pe primele $p = 6$ cartonașe se află 6 valori conform cerinței (1, 2, 3, 4, 5, 6).

5.2 Rezolvarea problemei Cartonașe

Cerința 1

Pentru cerința 1 este suficient să parcurgem elementele vectorului care preced poziția poz dată și să ne oprim la prima valoare care este mai mare decât $v[poz]$, afișând poziția respectivă. Dacă nu sunt astfel de valori înaintea poziției poz, atunci soluția va fi poz.

Cerința 2

Pentru a rezolva cerința 2, traversăm vectorul element cu element și la poziția curentă decidem dacă o afișăm sau nu.

O primă abordare este să parcurgem iarăși elementele de la început până la poziția curentă și să verificăm dacă toate sunt mai mici sau egale cu valoarea poziției curente. Timpul de calcul obținut este de ordinul $O(N^2)$ și această soluție nu se va încadra în timp pentru toate datele de test.

Ținând însă cont că toate elementele vectorului sunt distincte și au valori de la 1 la N, observăm că poziția curentă i este una care trebuie afișată dacă și numai dacă maximul din vector dintre elementele aflate până la poziția i este egal cu i (valoarea poziției curente).

Astfel, scriem un algoritm de calcul al maximului dintr-un vector, iar la poziția curentă i este suficient să-l afișăm pe i dacă și numai dacă maximul de până acum este i. Această abordare are timp de calcul de ordin $O(N)$.

Cerința 3

Soluția optimă pentru cerința 3 este asemănătoare cu cea descrisă pentru cerința 2. De data aceasta însă, la poziția curentă i este necesar să păstrăm atât maximul cât și al doilea maxim. Observăm că putem decide să afișăm poziția i dacă primul maxim este strict mai mare decât i și al doilea maxim este mai mic sau egal cu i. Avem, de asemenea, timp de calcul de ordin $O(N)$.

5.3 Cod-sursă pentru problema Cartonașe

```
#include <fstream>

#define DIM 100010
using namespace std;
int n, c, i, poz, maxim, maxim1, maxim2, nr, f[DIM], v[DIM];
int main () {
    ifstream fin ("cartonase.in");
    ofstream fout("cartonase.out");
    fin>>c>>n;
    for (i=1;i<=n;i++) {
        fin>>v[i];
        f[v[i]]++;
    }
    if (c == 1) {
        fin>>poz;
        for (i=1;i<=poz;i++)
            if (v[i] >= v[poz])
                break;
        fout<<i-1<<"\n";
        return 0;
    }

    if (c == 2) {
        maxim = 0;
        nr = 0;
        for (i=1;i<=n;i++) {
            if (v[i] > maxim)
                maxim = v[i];
            if (maxim == i) {
                nr++;
                if (nr != 1)
```

```
                    fout<<" ";
                fout<<i;
            }
        }
        fout<<"\n";
        return 0;
    }
    if (c == 3) {
        maxim1 = 0;
        maxim2 = 0;
        nr = 0;
        for (i=1;i<=n;i++) {
            if (v[i] > maxim1) {
                maxim2 = maxim1;
                maxim1 = v[i];
            } else
                if (v[i] > maxim2)
                    maxim2 = v[i];
            if (maxim1 > i && maxim2 <= i) {
                nr++;
                if (nr!=1)
                    fout<<" ";
                fout<<i;
            }
        }
        fout<<"\n";
    }
    return 0;
}
```

5.4 Problema Căsuțe

Propusă de: Dan-Constantin Spătărel, București

Există N căsuțe (pătrățele), așezate în ordine, de la stânga la dreapta, numerotate de la 1 la N. În interiorul fiecărei căsuțe putem scrie câte un număr natural. Inițial, în fiecare căsuță scriem același număr 0. Executăm, în ordine, Q operații, care pot fi de trei tipuri:

- Primul tip de operație se codifică prin **1 st dr nr** și înseamnă că în fiecare căsuță cu indicii între *st* inclusiv și *dr* exclusiv ștergem numerele care existau înainte și scriem în locul lor același număr *nr*.
- Al doilea tip de operație se codifică prin **2 poz** și rezultatul operației este numărul aflat în căsuța cu indicele *poz*.
- Al treilea tip de operație se codifică prin **3 st dr** și rezultatul operației este numărul de apariții al valorii celei mai mari din căsuțele cu indicii între *st* inclusiv și *dr* exclusiv.

Cerințe

Determinați rezultatele tuturor operațiilor de tip 2 sau 3, în ordinea executării acestora.

Date de intrare

Fișierul de intrare `casute.in` conține pe prima linie două numere naturale N și Q separate printr-un spațiu, cu semnificația din enunț. Pe fiecare dintre următoarele Q linii se află codificările celor Q operații.

Fiecare linie care codifică o operație începe cu un număr natural, reprezentând tipul operației, care poate fi 1, 2 sau 3 și este urmat de un spațiu.

- Dacă tipul operației este 1, atunci urmează trei numere naturale separate prin câte un spațiu: *st*, *dr* și *nr*, cu semnificația din enunț.
- Dacă tipul operației este 2, atunci urmează un singur număr natural *poz*, cu semnificația din enunț.
- Dacă tipul operației este 3, atunci urmează două numere naturale separate printr-un spațiu *st* și *dr*, cu semnificația din enunț.

Date de ieșire

Fișierul de ieșire `casute.out` conține, pentru fiecare operație de tip 2 sau 3, în ordinea în care acestea se regăsesc în fișierul de intrare, pe linii separate, câte un număr natural reprezentând rezultatul operației corespunzătoare.

Restricții

- $Q \leq 3000$;
- $N \leq 10^9$;
- $1 \leq \text{st} < \text{dr} \leq N + 1$ pentru orice operație de tipul 1 și 3;
- $1 \leq \text{poz} \leq N$ pentru orice operație de tipul 2;
- $1 \leq \text{nr} \leq 3000$ pentru orice operație de tipul 1;
- în tabelul de mai jos, notăm $Op = \{1, 2\}$ dacă există numai operații de tipul 1 și 2, sau $Op = \{1, 2, 3\}$ dacă există operații de toate tipurile (1, 2 și 3);
- în tabelul de mai jos, notăm $D = 1$ dacă oricare dintre valorile *st*, *dr* și *poz* apar într-o singură operație, sau $D = 0$ dacă se pot și repeta.

#	Puncte	Restricții		
1	25	$N \leq 3000,$	$D = 0,$	$Op = \{1, 2\}$
2	25	$N \leq 3000,$	$D = 0,$	$Op = \{1, 2, 3\}$
3	25	$N \leq 10^9,$	$D = 1,$	$Op = \{1, 2\}$
4	15	$N \leq 10^9,$	$D = 1,$	$Op = \{1, 2, 3\}$
5	10	$N \leq 10^9,$	$D = 0,$	$Op = \{1, 2, 3\}$

Exemple

casute.in	casute.out	Explicații
9 12 1 3 7 4 1 2 4 5 1 6 10 3 2 1 2 2 2 3 2 9 3 1 10 3 5 8 3 1 2 1 1 4 1 2 1	0 5 5 3 2 1 1 1	Sunt $N = 9$ căsuțe. Inițial numerele din cele 9 căsuțe sunt: 0 0 0 0 0 0 0 0 0 După prima operație: 1 3 7 4, numerele devin: 0 0 4 4 4 4 0 0 0 (scriem 4 pe pozițiile **3, 4, 5, 6**) După a doua operație: 1 2 4 5, numerele devin: 0 5 5 4 4 4 0 0 0 (scriem 5 pe pozițiile **2, 3**) După a treia operație: 1 6 10 3, numerele devin: 0 5 5 4 4 3 3 3 3 (scriem 3 pe pozițiile **6, 7, 8, 9**) Rezultatul pentru a patra operație: 2 1 este **0**. Rezultatul pentru a cincea operație: 2 2 este **5**. Rezultatul pentru a șasea operație: 2 3 este **5**. Rezultatul pentru a șaptea operație: 2 9 este **3**. Rezultatul pentru a opta operație: 3 1 10 este **2**, deoarece maximul din toate căsuțele este **5** iar acesta apare de două ori. Rezultatul pentru a noua operație: 3 5 8 este **1**, deoarece maximul din căsuțele cu valorile: **4 3 3** este **4** iar acesta apare o dată. Rezultatul pentru a zecea operație: 3 1 2 este **1**, deoarece maximul din căsuțele cu valorile: **0** este **0** iar acesta apare o dată. După a unsprezecea operație: 1 1 4 1, numerele devin: 1 1 1 4 4 3 3 3 3 (scriem 1 pe pozițiile **1, 2, 3**) Rezultatul pentru a doisprezecea operație: 2 1 este **1**.

5.5 Rezolvarea problemei Căsuțe

Cazurile 1 și 2 (50 de puncte)

Dacă $N \leq 3000$ atunci putem folosi un vector de dimensiune $N+1$ (deoarece vectorii sunt indexați de la 0, nu de la 1 ca în problemă) pentru a ține evidența numerelor naturale aflate în fiecare căsuță pe parcursul executării celor Q operații.

Cu ajutorul unui vector putem rezolva fiecare dintre cele 3 operații relativ ușor, astfel:

1. Operațiile de primul tip se rezolvă cu o atribuire în cadrul unei structuri repetitive de tip for.
2. Răspunsul pentru de al doilea tip de operație se obține accesând elementul de pe poziția poz din vector.
3. Răspunsul pentru de al treilea tip de operație se obține prin determinarea maximului și prin contorizarea numărului de apariții al acestuia în vector, în intervalul st inclusiv dr exclusiv.

Complexitatea spațiu: $O(N)$

Complexitatea timp: $O(Q \cdot N)$

Cazurile 1 și 3 (50 de puncte)

Avem de răspuns numai la operații de tip 2.

Observăm că rezultatul unei operații de tip 2 depinde doar de ultima operație de tip 1 care a afectat căsuța poz înainte de executarea operației de tip 2.

Deoarece $Q \leq 3000$ putem stoca toate operațiile, în ordinea în care trebuie executate (de exemplu cu ajutorul a 5 vectori cu denumirile: tip, st, dr, nr și poz).

Putem găsi răspunsul pentru fiecare operație de tip 2 astfel: căutăm operația anterioară, cea mai recentă, de tip 1, cu proprietatea: $st_1 \leq poz_2 < dr_1$ - răspunsul este nr_1. Dacă nu există nicio astfel de operație atunci răspunsul este 0.

Dacă observăm că există o echivalență între procesul de inițializare a căsuțelor cu valoarea 0 și operația fictivă $tip = 1$ $st = 1$ $dr = N + 1$ $nr = 0$, atunci o alternativă, pentru a evita cazul particular de mai sus, este să adăugăm, înainte de citirea operațiilor din fișierul de intrare, această operație.

Complexitatea spațiu: $O(Q)$

Complexitatea timp: $O(Q^2)$

Cazurile 1, 2 și 3 (75 de puncte)

Putem combina cele două soluții de mai sus, într-o singură sursă, astfel: Dacă $N \leq 3000$ atunci rezolvăm problema folosind prima soluție, altfel rezolvăm problema folosind a doua soluție.

Toate cazurile (100 de puncte)

Dacă $N = 10^9$, atunci orice fel de soluție care va încerca să rețină valorile din fiecare căsuță va obține verdictul limită de memorie depășită. Se impune astfel să economisim memoria utilizată.

Putem observa că toate cele N căsuțe pot fi împărțite în intervale maximale (care nu mai pot fi extinse) de căsuțe consecutive cu proprietatea că orice operație de tip 1:

- fie nu modifică niciuna dintre căsuțele din interval;
- fie modifică toate căsuțele din interval.

De aceea, dacă am identifica aceste intervale, am putea reține pentru fiecare dintre ele un singur număr natural: valoarea din fiecare dintre căsuțele din interval.

Să analizăm următorul exemplu: $N = 50$ și 4 operații de tipul 1 (inclusiv operația suplimentară echivalentă cu inițializarea):

Intervale	1...5	6...16	17...24	25...31	32...35	36...41	42...50	
index	1	6	17	25	32	36	42	51
tip=1 st=**1** dr=**51** nr=0	0	0	0	0	0	0	0	
tip=1 st=**6** dr=**42** nr=6	0	6	6	6	6	6	0	
tip=1 st=**25** dr=**36** nr=4	0	6	6	4	4	6	0	
tip=1 st=**17** dr=**32** nr=5	0	6	5	5	4	6	0	

Intervalele, vectorul index și valorile după fiecare dintre cele 4 operații

Observăm că toate intervalele pot fi descrise cu ajutorul vectorului *index* (descris în tabelul de mai sus) prin două elemente consecutive ale sale. Mai mult decât atât, observăm că elementele vectorului *index* sunt toate valorile *st* și *dr* de la toate operațiile de tip 1, sortate crescător.

În acest moment putem rezolva ușor operațiile de tip 2, identificând din ce interval face parte *poz*.

Operațiile de tip 3 sunt mai dificil de rezolvat, deoarece trebuie să identificăm atât intervalele care contribuie la rezultatul unei operații cât și lungimea intersecției dintre aceste intervale și intervalul operației.

O modalitate facilă de a simplifica rezolvarea, atât pentru operațiile de tip 3 cât și pentru cele de tip 2 este să adăugăm în plus la vectorul index atât valorile *st* și *dr* de la toate operațiile de tip 3 cât și valorile *poz* de la toate operațiile de tip 2. Acest proces va avea ca efect fragmentarea suplimentară a intervalelor, astfel încât ele își vor pierde proprietatea de maximalitate însă cu următoarele beneficii:

- valorile *poz* ale operațiilor de tip 2 se vor afla numai la începutul unui interval;
- intervalele formate sunt complet incluse în intervalul *st* inclusiv *dr* exclusiv al oricărei operații de tip 3.

Algoritm

Folosind tehnica descrisă în a doua soluție, vom stoca toate operațiile, în ordinea în care trebuie executate.

Într-un vector numit *index* vom pune laolaltă toate valorile *st*, *dr* și *poz* de la toate operațiile, inclusiv valorile speciale 1 și $N + 1$ corepunzătoare operației fictive $tip = 1$ $st = 1$ $dr = N + 1$ $nr = 0$, echivalentă cu procesul de inițializare a căsuțelor.

Vom sorta elementele vectorului *index* și apoi vom elimina dublurile (elementele care se repetă).

Similar cu prima soluție, vom folosi un vector de aceeași lungime ca și vectorul *index* pentru a stoca valorile celulelor din fiecare interval în parte.

Operațiile de primul tip se rezolvă astfel:

- cu ajutorul unei structuri repetitive se caută poziția elementului *st* în vectorul *index*;
- cu ajutorul unei structuri repetitive se caută poziția elementului *dr* în vectorul *index*;

- în cadrul celei de-a doua structuri repetitive se modifică vectorul de valori.

Răspunsul pentru de al doilea tip de operație se obține astfel:

- cu ajutorul unei structuri repetitive se caută poziția elementului *poz* în vectorul *index*;
- rezultatul operației se regăsește în vectorul de valori la poziția găsită la pasul anterior.

Răspunsul pentru de al treilea tip de operație se obține astfel:

- cu ajutorul unei structuri repetitive se caută poziția elementului *st* în vectorul index;
- cu ajutorul unei structuri repetitive se caută poziția elementului *dr* în vectorul index;
- în cadrul celei de-a doua structuri repetitive se calculează valoarea maximă din vectorul de valori;
- atunci când actualizăm maximul, resetăm numărul său de apariții la lungimea intervalului curent;
- de fiecare dată când găsim o nouă apariție maximului, creștem numărul său de apariții cu lungimea intervalului curent.

Operația de eliminare a dublurilor din vectorul *index* poate fi opțională. În funcție de detaliile de implementare, unele implementări pot lua punctaj maxim deși nu elimină dublurile.

Complexitatea spațiu: $O(Q)$

Complexitatea timp: $O(Q^2)$

5.6 Cod-sursă pentru problema Căsuțe

```
#include <fstream>
int main() {
  std::ifstream fisier_in("casute.in");
  std::ofstream fisier_out("casute.out");
  int N, Q;
  fisier_in >> N >> Q;
  int tip[Q];
  int st[Q];
  int dr[Q];
  int nr[Q];
  int poz[Q];

  int n = 0;
  int index[2 + 2 * Q];
  index[n++] = 1;
  index[n++] = N + 1;
  for (int i = 0; i < Q; i++) {
    fisier_in >> tip[i];
    if (tip[i] == 1) {
      fisier_in >> st[i] >> dr[i] >> nr[i];
      index[n++] = st[i];
      index[n++] = dr[i];
    } else if (tip[i] == 2) {
      fisier_in >> poz[i];
      index[n++] = poz[i];
    } else { // if (tip[i] == 3)
      fisier_in >> st[i] >> dr[i];
      index[n++] = st[i];
      index[n++] = dr[i];
    }
  }
  for (int i = 0; i < n; i++) {
```

```
    for (int j = i + 1; j < n; j++) {
      if (index[i] > index[j]) {
        int tmp = index[i];
        index[i] = index[j];
        index[j] = tmp;
      }
    }
  }
  int k = 1;
  for (int i = 1; i < n; i++) {
    if (index[i - 1] < index[i]) {
      index[k++] = index[i];
    }
  }
  n = k;
  int a[n];
  k = 0;
  while (index[k] < 1) {
    k++;
  }
  while (index[k] < N + 1) {
    a[k] = 0;  k++;
  }
  for (int i = 0; i < Q; i++) {
    if (tip[i] == 1) {
      k = 0;
      while (index[k] < st[i]) {
        k++;
      }
      while (index[k] < dr[i]) {
        a[k] = nr[i]; k++;
      }
    } else if (tip[i] == 2) {
      k = 0;
      while (index[k] < poz[i]) {
        k++;
      }
      fisier_out << a[k] << '\n';
    } else { // if (tip[i] == 3)
      k = 0;
      while (index[k] < st[i]) {
        k++;
      }
      int maxim = a[k], aparitii = 0;
      while (index[k] < dr[i]) {
        if (maxim < a[k]) {
          maxim = a[k]; aparitii = 0;
        }
        if (maxim == a[k]) {
          aparitii += index[k + 1] - index[k];
        }
        k++;
      }
      fisier_out << aparitii << '\n';
    }
  }
  return 0;
}
```

5.7 Problema Perechi

Propusă de: stud. Jonathan Mogovan, Universitatea „Babeș-Bolyai", Cluj-Napoca, Cluj

Gigel a primit o sarcină interesantă: se dă un șir de N numere numere naturale și un număr natural K.

Cerințe

1. Fie X primul număr din șir. Determinați poziția celui mai mic număr Y care aparține șirului, astfel încât suma celor două numere X și Y să fie divizibilă cu K. Dacă valoarea Y, cu proprietatea precizată, apare de mai multe ori în șir, se ia în considerare poziția cea mai din dreapta. Există cel puțin un astfel de număr Y, care aparține șirului.
2. Determinați numărul minim de elemente care trebuie eliminate din șir astfel încât elementele rămase să poată fi grupate în perechi disjuncte (fiecare element rămas aparține unei singure perechi), cu proprietatea că suma celor două valori din fiecare pereche este divizibilă cu K.

Date de intrare

Fișierul de intrare `perechi.in` conține:

- pe prima linie, un număr natural C reprezentând cerința de rezolvat (1 sau 2);
- pe cea de-a doua linie, două numere naturale N și K, cu semnificația din enunț;
- pe cea de-a treia linie, N numere naturale, reprezentând elementele șirului.

Numerele aflate pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `perechi.out` conține, pe prima linie, un număr natural, reprezentând numărul determinat conform cerinței C.

Restricții

- $2 \leq N \leq 10^5$;
- $1 \leq K \leq 10^5$;
- toate elementele șirului au valori cuprinse între 0 și 10^9;
- pentru $C = 1$, poziția primului element X nu coincide cu poziția lui Y;
- o pereche este formată din exact două elemente.

#	Puncte	Restricții
1	31	$C = 1$
2	69	$C = 2$

Exemple

perechi.in	perechi.out	Explicații
1 7 3 2 3 4 5 1 1 2	6	$C = 1$, $N = 7$, $K = 3$, șirul este $[2, 3, 4, 5, 1, 1, 2]$, iar $X = 2$. Valorile lui Y din șir pentru care $(X + Y)\%3 = 0$ sunt: 4 (poziția 3, deoarece $2 + 4 = 6$) și 1 (pozițiile 5 și 6, deoarece $2 + 1 = 3$). Astfel, valoarea minimă cerută cu proprietatea precizată este $Y = 1$, iar cea mai din dreapta poziție a sa este 6.
2 4 4 1 2 3 4	2	$C = 2$, $N = 4$, $K = 4$, șirul este $[1, 2, 3, 4]$. Dacă eliminăm elementele 2 și 4, rămân 1 și 3, care formează o pereche cu suma $1 + 3 = 4$, divizibilă cu 4. Răspunsul este 2
2 6 2 2 4 6 8 10 12	0	$C = 2$, $N = 6$, $K = 2$, șirul este $[2, 4, 6, 8, 10, 12]$. Se pot forma perechile (2, 4), (6, 8), (10, 12), cu sumele 6, 14, 22, fiecare divizibilă cu 2. O alta modalitate de a forma perechi este: (2, 8), (4, 10), (6, 12), cu sumele 10, 14, 18, fiecare divizibilă cu 2. Astfel, răspunsul este 0.

5.8 Rezolvarea problemei Perechi

Cerința 1

Pentru cerința 1 este suficient să găsim cel mai din dreapta element care respectă condiția. Parcurgem șirul și identificăm cel mai mic număr Y pentru care suma $X + Y$ este divizibilă cu K, alegând poziția celui mai din dreapta Y. Complexitate: $O(N)$.

Cerința 2

Pentru cerința 2, o abordare naivă verifică toate perechile posibile în două bucle și marchează valorile din perechile valide pentru a nu le reutiliza. Complexitatea de timp este: $O(N^2)$.

Soluția optimă împarte numerele în funcție de resturile lor la K cu ajutorul unui vector de frecvență și le grupează în perechi de forma $(R,\ K - R)$, unde $1 \leq R \leq K/2$. Se analizează separat cazurile pentru restul 0 și pentru K par. Pentru fiecare pereche de forma de mai sus, adunăm la rezultat diferența dintre valoarea frecvenței mai mari și valoarea frecvenței mai mici. Complexitate: $O(N + K)$.

5.9 Cod-sursă pentru problema Perechi

```
#include <fstream>
using namespace std;
ifstream fin("perechi.in");
ofstream fout("perechi.out");
```

```
int n, x, a, b, c, y, k, i, poz, fr[100001];
int main()
{
    fin >> c >> n >> k;
    a = 1000000001;
    if (c == 1)
    {
        fin >> x;
        if (x % k ==0)
        {
            a = x;
            poz = 1;
        }
        for (i = 2; i <= n; i++)
        {
            fin >> y;
            if ( ((x % k) + (y % k)) % k == 0 || y % k == 0)
            {
                if (y <= a)
                {
                    a = y;
                    poz = i;
                }
            }
        }
        fout << poz;
    }
    else
    {
        for (i = 1; i <= n; i++)
        {
            fin >> x;
            fr[x % k]++;
        }
        int nr = 0;
        if (fr[0] % 2 == 1)  nr++;
        if (k % 2 == 1)
            x = k + 1;
        else
            x = k;
        for (i = 1; i < x/2; i++)
        {
            if (i % k != 0)
            {
                a = max (fr[i], fr[k-i]);
                b = min (fr[i], fr[k-i]);
                nr = nr + a - b;
            }
        }
        if (k % 2 == 0) nr= nr + (fr[k/2] % 2);
        fout << nr;
    }
    return 0;
}
```

Capitolul 6

ONI 2025, clasa a VI-a

6.1 Problema Diff

Propusă de: prof. Dan Pracsiu, Liceul Teoretic „Emil Racoviță”, Vaslui

Se consideră șirul de N cifre nenule $a = (a_1, a_2, ..., a_N)$. Prin frecvență de apariție a unei cifre în șir înțelegem numărul de apariții ale cifrei în acest șir.

Pentru o secvență $a_i, a_{i+1}, ..., a_j$ din acest șir ($1 \leq i < j \leq N$) calculăm frecvența fiecărei cifre distincte prezente în secvență și definim **diff**-ul secvenței ca fiind diferența dintre cea mai mare frecvență și cea mai mică frecvență dintre cele calculate.

Exemplul 1: în secvența $2, 7, 3, 2, 2, 3, 8, 8, 2$ **diff**-ul secvenței este $4 - 1 = 3$ (cifra 2 apare de patru ori, iar cifra 7 o singură dată).

Exemplul 2: pentru secvența $9, 9, 9, 9$ **diff**-ul secvenței este 0.

Cerințe

1. Determinați frecvența maximă de apariție a unei cifre din șirul a.
2. Determinați **diff**-ul maxim posibil al unei secvențe care începe de la prima poziție din șirul a.
3. Determinați **diff**-ul maxim al unei secvențe din șirul a.

Date de intrare

Fișierul de intrare `diff.in` conține pe prima linie numerele naturale C și N, unde C este cerința care trebuie rezolvată (1, 2 sau 3) și N are semnificația din enunț, iar pe următoarea linie N cifre nenule, separate prin câte un spațiu, reprezentând termenii șirului a.

Date de ieșire

Fișierul de ieșire `diff.out` conține numărul determinat pentru cerința C.

Restricții

- $C \in \{1, 2, 3\}$
- $3 \leq N \leq 100\,000$
- Se garantează că, pentru toate testele, în șir există cel puțin două cifre distincte.

#	Puncte	Restricții
1	30	$C = 1$
2	30	$C = 2$
3	40	$C = 3$

Exemple

diff.in	diff.out	Explicații
1 9 1 7 7 9 7 7 1 9 1	4	$C = 1$, $N = 9$. Se rezolvă cerința 1. Șirul $a = (1, 7, 7, 9, 7, 7, 1, 9, 1)$ conține cifra 1 de 3 ori, cifra 7 de 4 ori, cifra 9 de 2 ori. Frecvența maximă de apariție este 4, corespunzătoare cifrei 7.
2 9 1 7 7 9 7 7 1 9 1	3	$C = 2$, $N = 9$. Se rezolvă cerința 2. **diff**-ul maxim al unei secvențe care începe de la poziția 1 este 3 și aparține secvenței: 1 7 7 9 7 7
3 10 9 7 7 9 7 7 9 7 7 9	4	$C = 3$, $N = 10$. Se rezolvă cerința 3. **diff**-ul maxim este 4, corespunzător secvenței: 7 7 9 7 7 9 7 7

6.2 Rezolvarea problemei Diff

Cerința 1 - 30p

Se utilizează un vector de frecvențe, notat cu fr, de lungime 10, în care $fr[i]$ reține numărul de apariții ale cifrei i în șirul a, $i = 1..9$. Parcurgem șirul a și fiecare cifră $a[i]$ o contorizăm în fr, iar la final aflăm valoarea maximă din fr.

Cerința 2 - 30p

Ca și la prima cerință, utilizăm vectorul fr. Parcurgem șirul a și la fiecare pas $i = 1..n$, îl punem pe $a[i]$ în vectorul de frecvențe. În acest moment în fr avem memorat numărul de apariții ale fiecărei cifre de la 1 la 9 din secvența $a[1]$, $a[2]$, ..., $a[i]$. Actualizăm diferența maximă dintre două valori nenule din fr.

Cerința 3 - 40p. Soluția 1

Avem două etape:

1. Să considerăm pentru început două cifre distincte $c1$ și $c2$. Dorim să determinăm diff-ul maxim (diferența maximă dintre numărul de apariții ale lui $c1$, minus numărul de apariții ale lui $c2$) care se obține dintr-o secvență din șir.
   Construim un vector d de lungime n.
   Parcurgem șirul a și la fiecare pas i avem cazurile:
   - $a[i] = c1$, atunci $d[i] = d[i-1] + 1$
   - $a[i] = c2$, atunci $d[i] = d[i-1] - 1$
   - $a[i] \neq c1$ și $a[i] \neq c2$, atunci $d[i] = d[i-1]$

Deci valorile din d cresc atunci când dăm peste cifra $c1$, scad când dăm peste $c2$ sau rămân nemodificate în cazul celorlalte litere. La fiecare pas i, vom reține în variabila mn cea mai mică valoare a lui d care s-a obținut până atunci când s-a găsit o cifră $c2$. Diferența maximă diff se actualizează cu valoarea $d[i] - mn$ de fiecare dată când întâlnim o cifră $c1$.
Atenție, valorile din d pot fi și negative sau zero, dar trebuie să ne asigurăm că valoarea mn s-a obținut atunci când am întâlnit cel puțin un $c2$.

2. Facem același algoritm pentru cele două cifre $c1$ și $c2$, parcurgând șirul de la dreapta la stânga.

Etapele 1 și 2 le efectuăm pentru orice cifre diferite, $1 \leq c1 \neq c2 \leq 9$. Numărul de pași la cerința 3 va fi deci $9 * 8 * n = 72 * n$.

Cerința 3 - 40p. Soluția 2

Rezolvarea cerinței este bazată pe algoritmul de subsecvență de sumă maximă (Kadane).

Pentru fiecare pereche de cifre $c1$ și $c2$, $1 \leq c1, c2 \leq 9$, $c1 \neq c2$ parcurgem șirul a și pentru fiecare element a_i al șirului vom defini variabila x astfel: 1 dacă $a_i = c1$, -1 dacă $a_i = c2$ sau 0 în caz contrar.

Vom adăuga valoarea x la suma subsecvenței curente, sumă ce reprezintă de fapt **diff**-ul maxim al subsecvenței pentru care cifra $c1$ are numărul de apariții maxim, iar cifra $c2$ are numărul de apariții minim.

Evident, vom reține valoarea maximă a sumei secvenței doar dacă cifra $c2$ apare.

6.3 Cod-sursă pentru problema Diff

```
#include <bits/stdc++.h>
using namespace std;

ifstream  fin("diff.in");
ofstream fout("diff.out");
int fr[10], n, a[100006];
int d[100006];

int main()
{
    int task, i, j, c1, c2, difmax, minAp, maxAp;
    int ult, mn;
    fin >> task >> n;
    for (i = 1; i <= n; i++)
        fin >> a[i];
    if (task == 1)
    {
        int mx = 0;
        for (i = 1; i <= n; i++)
        {
            j = a[i];
            fr[j]++;
            mx = max(mx, fr[j]);
        }
        fout << mx << "\n";
    }
    else if (task == 2)
    {
        difmax = 0;
```

```
        for (i = 1; i <= n; i++)
        {
            fr[a[i]]++;
            minAp = 100001; maxAp = 0;
            for (j = 1; j < 10; j++)
                if (fr[j] > 0)
                {
                    minAp = min(minAp, fr[j]);
                    maxAp = max(maxAp, fr[j]);
                }
            difmax = max(difmax, maxAp - minAp);
        }
        fout << difmax << "\n";
    }
    else /// task == 3
    {
        difmax = 0;
        for (c1 = 1; c1 < 9; c1++)
            for (c2 = c1 + 1; c2 < 10; c2++)
            {
                ///---------------------------------------
                ult = mn = 0;
                for (i = 1; i <= n; i++)
                {
                    if (a[i] == c1)
                    {
                        d[i] = d[i - 1] + 1;
                        if (ult > 0) difmax = max(difmax, d[i] - mn);
                    }
                    else if (a[i] == c2)
                    {
                        d[i] = d[i - 1] - 1;
                        for (j = ult; j < i; j++)
                            if (mn > d[j]) mn = d[j];
                        ult = i;
                    }
                    else d[i] = d[i - 1];
                }
                ult = n + 1; mn = 0;
                for (i = n; i >= 1; i--)
                {
                    if (a[i] == c1)
                    {
                        d[i] = d[i + 1] + 1;
                        if (ult <= n) difmax = max(difmax, d[i] - mn);
                    }
                    else if (a[i] == c2)
                    {
                        d[i] = d[i + 1] - 1;
                        for (j = ult; j > i; j--)
                            if (mn > d[j]) mn = d[j];
                        ult = i;
                    }
                    else d[i] = d[i + 1];
                }

                swap(c1, c2);
                ult = mn = 0;
                for (i = 1; i <= n; i++)
                {
                    if (a[i] == c1)
```

```
                    {
                        d[i] = d[i - 1] + 1;
                        if (ult > 0) difmax = max(difmax, d[i] - mn);
                    }
                    else if (a[i] == c2)
                    {
                        d[i] = d[i - 1] - 1;
                        for (j = ult; j < i; j++)
                            if (mn > d[j]) mn = d[j];
                        ult = i;
                    }
                    else d[i] = d[i - 1];
                }
                ult = n + 1; mn = 0;
                for (i = n; i >= 1; i--)
                {
                    if (a[i] == c1)
                    {
                        d[i] = d[i + 1] + 1;
                        if (ult <= n) difmax = max(difmax, d[i] - mn);
                    }
                    else if (a[i] == c2)
                    {
                        d[i] = d[i + 1] - 1;
                        for (j = ult; j > i; j--)
                            if (mn > d[j]) mn = d[j];
                        ult = i;
                    }
                    else d[i] = d[i + 1];
                }
                swap(c1, c2);
                ///-----------------------------------------
            }
        fout << difmax << "\n";
    }
    return 0;
}
```

```
#include <fstream>
const int NMAX=1e5+5;
const int CMAX=15;

using namespace std;
ifstream cin("diff.in");
ofstream cout("diff.out");

int a[NMAX], f[CMAX];
int n;

int main()
{
    int c, i, j, ans=0, cnt=0;
    cin>>c>>n;
    for(i=1; i<=n; i++)
    {
        cin>>a[i];
        if(!f[a[i]]) cnt++;
        f[a[i]]++;
        ans=max(ans, f[a[i]]);
    }
    if(c!=1)
    {
```

```
        ans=0;
        for(i=1; i<=9; i++)
        {
            for(j=1; j<=9; j++)
            {
                if(i==j) continue;
                int k, minim=1e9, minrev=0, sum=0;
                for(k=1; k<=n; k++)
                {
                    if(a[k]==i) sum++;
                    else if(a[k]==j)
                    {
                        sum--;
                        minim=min(minim, minrev);
                    }
                    if(c==3) ans=max(ans, sum-minim);
                    else if(minim!=1e9) ans=max(ans, sum);
                    minrev=min(minrev, sum);
                }
            }
        }
    }
    cout<<ans<<'\n';
    return 0;
}
```

6.4 Problema Prime

Propusă de: prof. Ionel-Vasile Piț-Rada, Colegiul Național „Traian" Drobeta-Turnu Severin

Pentru un număr natural N considerăm șirul: 0, 1, 2, ..., N.

Cerințe

1. Se dau Q perechi de numere naturale de forma (a, b). Pentru fiecare pereche se cere să se determine numărul de numere prime care află în secvența de numere consecutive: a, $a+1$, $a+2$, ..., b.
2. Se dau Q numere naturale p_1, p_2, ..., p_Q. Pentru fiecare număr p_i se cere să se determine numărul secvențelor a, $a+1$, $a+2$, ..., b din șirul 0, 1, ..., N care conțin câte p_i numere prime ($1 \leq i \leq Q$).

Date de intrare

Fișierul de intrare `prime.in` conține pe prima linie trei numere naturale C N Q, separate prin câte un spațiu, unde C este cerința care trebuie rezolvată (1 sau 2), N și Q au semnificația de mai sus.

Dacă $C = 1$, atunci pe fiecare dintre următoarele Q linii se află câte două numere naturale a b, separate prin spațiu, reprezentând extremitățile unei secvențe de numere naturale consecutive.

Dacă $C = 2$, atunci pe următoarele Q linii se află câte un număr natural p_i ($1 \leq i \leq Q$), cu semnificația din enunț.

Date de ieșire

Fișierul de ieșire `prime.out` conține Q numere, fiecare pe câte un rând, în conformitate cu cerința C.

Restricții

- $C \in \{1, 2\}$
- $1 \leq N, Q \leq 50\,000$
- $0 \leq a \leq b \leq N$
- $0 \leq p_i \leq N$, $1 \leq i \leq Q$

#	Puncte	Restricții
1	40	$C = 1$, $1 \leq N, Q \leq 10\,000$
2	10	$C = 1$, $10\,000 < N, Q \leq 50\,000$
3	30	$C = 2$, $1 \leq N, Q \leq 10\,000$
4	20	$C = 2$, $10\,000 < N, Q \leq 50\,000$

Exemple

prime.in	prime.out	Explicații
1 10 3 0 10 3 3 8 10	4 1 0	$C = 1$, $N = 10$, $Q = 3$. Se rezolvă cerința 1. În secvența $0 \ldots 10$ există 4 numere prime: 2, 3, 5, 7. În secvența $3 \ldots 3$ există un singur număr prim, numărul 3. În secvența $8 \ldots 10$ nu există numere prime.
2 10 2 4 1	12 17	$C = 2$, $N = 10$, $Q = 2$. Se rezolvă cerința 2. Există câte 4 numere prime în fiecare dintre următoarele 12 secvențe: $0 \ldots 10$, $1 \ldots 10$, $2 \ldots 10$, $0 \ldots 9$, $1 \ldots 9$, $2 \ldots 9$, $0 \ldots 8$, $1 \ldots 8$, $2 \ldots 8$, $0 \ldots 7$, $1 \ldots 7$, $2 \ldots 7$. Există câte un singur număr prim în fiecare dintre următoarele 17 secvențe: $0 \ldots 2$, $1 \ldots 2$, $2 \ldots 2$, $3 \ldots 3$, $3 \ldots 4$, $4 \ldots 5$, $5 \ldots 5$, $4 \ldots 6$, $5 \ldots 6$, $6 \ldots 7$, $6 \ldots 8$, $6 \ldots 9$, $6 \ldots 10$, $7 \ldots 7$, $7 \ldots 8$, $7 \ldots 9$, $7 \ldots 10$.

6.5 Rezolvarea problemei Prime

Soluția 1

Cerința 1. Varianta 1-1 - $O(Q * N * sqrt(N))$

Parcurgem cele Q interogări și pentru fiecare pereche a, b vom număra numerele prime din secvența $a, a + 1, ..., b$. Punctajul obținut va depinde de modul cum verificăm primalitatea unui număr.

Cerința 1. Varianta 1-2 - $O(N * sqrt(N) + Q * N)$

Verificăm primalitatea fiecărui număr din secvența $0...N$ și păstrăm informațiile într-un vector $v[i] = 1$, dacă i este prim, respectiv $v[i] = 0$ dacă i nu este prim. Parcurgem cele Q interogări și pentru fiecare pereche a, b vom număra numerele prime din secvența $a, a + 1, ..., b$.

Cerința 1. Varianta 1-3 - $O(N * log(N) + Q * N)$

Utilizăm ciurul lui Eratostene și construim vectorul $ciur[i] = 0$, dacă i este prim, respectiv $ciur[i] = 1$, în caz contrar. Parcurgem cele Q interogări și pentru fiecare pereche a, b vom număra numerele prime din secvența $a, a + 1, ..., b$ însumând valorile.

$(1 - ciur[a]) + (1 - ciur[a + 1]) + ... + (1 - ciur[b])$

Cerința 1. Varianta 1-4 - $O(N * log(N) + Q)$

Utilizăm ciurul lui Eratostene calculăm vectorul de sume parțiale $s[0...N]$. Parcurgem cele Q interogări și pentru fiecare pereche a, b vom număra numerele prime din secvența $a, a + 1, ..., b$ astfel: dacă $a = 0$, atunci numărul de numere prime este $s[b]$, în caz contrar $(a > 0)$ numărul de numere prime este $s[b] - s[a - 1]$.

Cerința 2. Varianta 2-1 - $O(Q * N * N * sqrt(N))$

Parcurgem cele Q interogări și în cadrul fiecărei interogări parcurgem toate secvențele $[a..b]$ cu $0 <= a <= b <= N$ și numărăm numerele prime în cadrul fiecărei secvențe.

Punctajul obținut va depinde de modul cum verificăm primalitatea unui număr și de modul cum efectuăm numărarea.

Cerința 2. Varianta 2-2 - $O(N * log(N) + Q * N * N)$

Îmbunătățim varianta $2 - 1$ utilizând ciurul lui Eratostene.

Cerința 2. Varianta 2-3 - $O(N * log(N) + N * N + Q)$

Îmbunătățim varianta $2 - 2$ astfel încât să facem doar o singură dată parcurgerea secvențelor. Pentru aceasta folosim un vector de frecvență ce calculează numărul de secvențe cu p numere prime.

Parcurgem cele Q interogări și pentru fiecare interogări afișăm corespunzător frecvența cerută.

Cerința 2. Varianta 2-4

Utilizând ciurul lui Eratostene calculăm vectorul $prime[]$ cu cele nr numere prime cuprinse în $[0..N]$, adăugăm vectorului elementele $prime[0] = -1$ și $prime[nr + 1] = N + 1$.

Se observă că pentru interogările în care se cere numărul secvențelor care nu conțin numere prime soluția se obține dacă numărăm pentru fiecare interval de valori $prime[i] + 1$,...,$prime[i + 1] - 1$ câte secvențe $[a..b]$ avem cu $prime[i] + 1 <= a <= b <= prime[i + 1] - 1$, $0 <= i <= nr$.

Notăm cu $x = prime[i + 1] - prime[i] - 1$

Formula este $1 + 2 + ... + x = x * (x + 1)/2$ (obținută cu formula lui Gauss)

Pentru interogările în care se cere numărul secvențelor care conțin p numere prime, cu $p >= 1$, soluția este să calculăm numărul de secvențe $[a..b]$ cu $prime[i - 1] + 1 <= a <= prime[i]$ si $prime[i + p - 1] <= b <= prime[i + p] - 1$, cu $0 <= i <= nr + 1 - p$.

Vom nota cu $x = prime[i] - prime[i - 1]$ și cu $y = prime[i + p] - prime[i + p - 1]$

Formula acum se reduce la $x * y$

Trebuie însumate aceste produse pentru a obține rezultatul, care se memorează pentru a nu mai fi recalculat

$O(N * log(N) + pi(N) * min(Q, pi(N)))$, unde $pi(N)$ reprezintă numărul numerelor prime $<= N$.

Soluția 2

Construim cu ciurul lui Eratostene un vector caracteristic a, de lungime 50000, în care $a[i] = 1$, dacă i este număr prim sau $a[i] = 0$, dacă i nu este număr prim. Pe baza acestui vector construim apoi sumele parțiale sp, deci $sp[i] = a[0] + a[1] + \cdots + a[i]$.

Cerința 1 Complexitate O(N*log(log(N))+Q)

Pentru fiecare întrebare dată prin perechea (i, j), numărul de numere prime din intervalul $[i, j]$ este dat de $sp[j] - sp[i - 1]$. Atenție, dacă $i = 0$, atunci răspunsul este $sp[j] - sp[0]$.

Cerința 2 Complexitate $O(N * log(log(N)) + N * pi(N)) + Q)$, unde $pi(N)$ reprezintă numărul numerelor prime $<= N$

Pentru a înțelege mai bine algoritmul la această cerință, să vedem cum arată vectorii a și sp pentru numerele de la 0 la 16:

i	=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a	=	0	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0	0
sp	=	0	0	1	2	2	3	3	4	4	4	4	5	5	6	6	6	6

Rețineți că cei doi vectori au de fapt lungimea 50000. Construim (precalculăm) încă doi vectori, fr și cnt, în care $fr[i]$ = câte valori din sp sunt egale cu i $cnt[i]$ = câte secvențe au exact i numere prime

Vectorul fr se construiește ușor, parcurgând sp. Cum construim însă pe cnt? Presupunem că suntem la un pas $i = 1..n$ și considerăm $x = sp[i]$.

Câte secvențe care se termină cu poziția i conțin zero numere prime? Răspunsul este dat de numărul de valori egale cu x obținute anterior. Să ne uităm la vectorul a obținut mai sus. La poziția 10 avem $x = sp[10] = 4$. Anterior mai avem încă trei de 4, deci sunt trei intervale care se termină cu 10 și au zero numere prime: $[8, 10]$, $[9, 10]$ și $[10, 10]$. Deci în $cnt[0]$ vom adăuga numărul de valori de x reținute anterior în vectorul fr.

Asemănător, pentru $x = sp[i]$, câte secvențe care se termină cu poziția i și au de exemplu două numere prime? Răspunsul este $fr[x - 2]$, care se adaugă la $cnt[2]$.

Ideea este deci că la fiecare pas, pentru $x = sp[i]$, pentru orice $j = 0..x$ putem contoriza câte intervale care se termină cu i au exact j numere prime, contorizând în $cnt[j]$ valoarea $fr[x - j]$. Nu uităm ca la final să-l adăugăm în fr pe x.

6.6 Cod-sursă pentru problema Prime

```
#include <fstream>
using namespace std;
ifstream  fin("prime.in");
ofstream fout("prime.out");
int N,Q,C,a,b,p,nv,v[50002],sp[50002];
long long T[50002];
char ciur[50002];
int main(){
    fin>>C>>N>>Q;
    ///eratostene
    ciur[0]=1; ciur[1]=1;
    for(int i=2;i<=N;i++){
        if(ciur[i]==0){
            int x=N/i;
            for(int j=i;j<=x;j++){
                ciur[j*i]=1;
            }
        }
    }
    if(C==1){
        ///sume partiale
        sp[0]=0;
        for(int i=1;i<=N;i++){
            sp[i]=sp[i-1]+(1-ciur[i]);
        }
```

```
    }
    if(C==2){
        ///numere prime
        nv=0;
        for(int i=2;i<=N;i++){
            if(ciur[i]==0){
                v[++nv]=i;
            }
        }
    }
    for(int q=1;q<=Q;q++){
        if(C==1){
            fin>>a>>b;
            if(a==0){
                fout<<sp[b]<<"\n";
            }
            else{
                fout<<sp[b]-sp[a-1]<<"\n";
            }
        }
        if(C==2){
            fin>>p;
            if(T[p]==0){
                long long k2=0;
                if(p==0){
                    v[0]=-1; v[nv+1]=N+1;
                    for(int i=1;i<=nv+1;i++){
                        long long a=v[i]-v[i-1]-1;
                        k2=k2+(a+1)*a/2;
                    }
                }
                else{
                    v[0]=-1; v[nv+1]=N+1;
                    for(int i=1;i+p-1<=nv;i++){
                        long long a=v[i]-v[i-1];
                        long long b=v[i+p]-v[i+p-1];
                        k2=k2+a*b;
                    }
                }
                T[p]=k2;
            }
            fout<<T[p]<<"\n";
        }
    }
    return 0;
}
```

6.7 Problema Special

Propusă de: prof. Ana-Maria Arișanu, Colegiul Național „Mircea cel Bătrân”, Râmnicu-Vâlcea

Mihai și Ioana au creat o reprezentare a matricii A cu N linii (numerotate de la 0 la $N-1$) și M coloane (numerotate de la 0 la $M-1$) în care fiecare element $A[i][j]$ este determinat pe baza următoarei formule: $A[i][j] = (15 * i + 4 * j + 2025)\%K$, unde i și j sunt indicii liniei și coloanei, iar K este un număr natural nenul, ales de ei.

Se definesc următoarele categorii de numere:

- **număr special**: un număr natural de două cifre, al cărui pătrat este un număr de trei cifre, iar cifra zecilor din acest pătrat este egală cu suma dintre cifra sutelor și cifra unităților. Exemplu: 11 este un număr special.
- **număr aproape special**: un număr care poate deveni special prin eliminarea a cel puțin unei cifre. Exemplu: 12310 este număr aproape special pentru că prin eliminarea cifrelor 0, 2 și 3 se obține numărul special 11.

Mihai și Ioana încep, în același timp și cu aceeași viteză, explorarea matricii începând cu $A[0][0]$, folosind strategii diferite:

- Ioana se deplasează pe linii, de sus în jos, și pe fiecare linie de la stânga la dreapta.
- Mihai se deplasează pe coloane, de la stânga la dreapta, și pe fiecare coloană de sus în jos.

În anumite momente de timp, cei doi ajung simultan la același element.

Cerințe

1. Determinați numărul de numere speciale care există în matricea A.
2. Determinați numărul elementelor din matricea A care sunt numere aproape speciale, la care Mihai și Ioana ajung în același timp.

Date de intrare

Fișierul de intrare `special.in` conține pe prima linie numărul natural C, unde C este cerința care trebuie rezolvată (1 sau 2). Pe a doua linie se află trei numere N M K, separate prin câte un spațiu, cu semnificația din enunț.

Date de ieșire

Fișierul de ieșire `special.out` conține numărul determinat pentru cerința C.

Restricții

- $C \in \{1, 2\}$
- $1 \leq N, M \leq 1\,000\,000$
- $10 \leq K \leq 1\,000\,000$

#	Puncte	Restricții
1	35	$C = 1$, $1 \leq N, M \leq 1\,000$
2	15	$C = 1$, $1\,000 < N, M \leq 1\,000\,000$
3	30	$C = 2$, $1 \leq N, M \leq 1\,000$
4	20	$C = 2$, $1\,000 < N, M \leq 1\,000\,000$

Exemple

special.in	special.out	Explicații
1 4 5 13	2	$C = 1$. Se rezolvă cerința 1. Matricea este 10 1 5 9 0 12 3 7 11 2 1 5 9 0 4 3 7 11 2 6 Numărul 11 este special și apare de 2 ori.
2 7 5 1000	1	$C = 2$. Se rezolvă cerința 2. Matricea este 25 29 33 37 41 40 44 48 52 56 55 59 63 67 71 70 74 78 82 86 85 89 93 97 101 100 104 108 112 116 115 119 123 127 131 Cei doi copii se întâlnesc la plecare în $A[0][0]$, apoi în $A[3][2]$ și la sosire în $A[6][4]$. În $A[6][4]$ se află numărul 131 care este aproape special. Ioana $\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 \\ 30 & 31 & 32 & 33 & 34 \end{bmatrix}$ Mihai $\begin{bmatrix} 0 & 7 & 14 & 21 & 28 \\ 1 & 8 & 15 & 22 & 29 \\ 2 & 9 & 16 & 23 & 30 \\ 3 & 10 & 17 & 24 & 31 \\ 4 & 11 & 18 & 25 & 32 \\ 5 & 12 & 19 & 26 & 33 \\ 6 & 13 & 20 & 27 & 34 \end{bmatrix}$
2 11 21 3000	5	$C = 2$. Se rezolvă cerința 2. Sunt 11 elemente ale matricii în care cei doi copii se întâlnesc, dintre care doar 5 au ca valori numere aproape speciale.

6.8 Rezolvarea problemei Special

Considerații matematice

Numărul natural $\overline{ab}$ este special $\iff \overline{ab}^2 = \overline{xyz} \iff y = x + z \implies \overline{ab}^2 = 100 \cdot x + 10 \cdot (x + z) + z \implies \overline{ab}^2 = 110 \cdot x + 11 \cdot z \implies \overline{ab}^2 = 11 \cdot \overline{xz} \implies \overline{ab} \in \{11, 22, 33\}$

$\overline{ab} = 11 \implies \overline{ab}^2 = 121$

$\overline{ab} = 22 \implies \overline{ab}^2 = 485$

$\overline{ab} = 33 \implies \overline{ab}^2 = 1089$ care nu convine pentru că nu are 3 cifre.

Deci, doar 11 și 22 pot fi considerate numere speciale.

Un număr devine număr special prin eliminarea cel puțin a unei cifre dacă numărul are cel puțin 3 cifre și conține cel puțin două cifre de 1 sau cel puțin două cifre de 2.

`Observație` Din cauza restricțiilor impuse soluția optimă nu poate fi obținută prin construirea efectivă a matricii.

Cerința 1. Soluție de $35p$ - $O(N \times M)$

Se generează toate elementele matricii și se verifică, pentru fiecare număr format din două cifre, dacă acesta îndeplinește condiția de „număr special”.

Cerința 1. Soluție de $43p$ - $O(N \times M)$

Se calculează frecvența de apariție a numerelor 11 și 22, conform formulei date.

Cerința 1. Soluție de $50p$ - complexitate $O(N + M)$

Propusă de: prof. Ionel-Vasile Piț Rada, Colegiul Național „Traian”, Drobeta-Turnu Severin

Se observă că, pe fiecare linie a matricii variația rezultatului formulei de calcul este dată de termenul $4 * j$ (deoarece $15 * i + 2025$ rămâne constant pe linie).

Aplicând proprietatea modulo $(a + b)\%k = (a\%k + b\%k)\%k$ se precalculează resturile termenilor $(4 * j)\%k$, pentru j variind de la 0 la $M - 1$.

Pentru fiecare linie i, valoarea $x = (15 * i + 2025)\%k$ este constantă și ca urmare numerele 11, respectiv 22 nu pot apărea decât în coloanele j pentru care $(4 * j)\%k = 11 - x$, $(4 * j)\%k = 22 - x$ sau $(4 * j)\%k = (k - x + 11)$, $(4 * j)\%k = (k - x + 22)$.

Cerința 2. Soluție de $30p$ - $O(N \times M)$

Se simulează deplasările construind matricile timpilor de acces la celule și se determină pe baza acestora punctele de întâlnire. Se contorizează doar punctele de întâlnire care au numere de cel puțin 3 cifre și care conțin fie cel puțin două cifre de 1, fie cel puțin două cifre de 2 (sau ambele).

Cerința 2. Soluție de $50p$ - $O(max(gcd(N, M), log(min(N, M))))$

Propusă de: prof. Alice Georgescu Alice, Colegiul Național „Mihai Viteazul”, Ploiești

Un punct de întâlnire (lin, col) are următoarea proprietate într-o matrice de dimensiune $N \times M$ (indexată de la 0) $lin \cdot M + col = col \cdot N + lin \implies lin \cdot (M - 1) = col \cdot (N - 1) \implies lin/col = (N - 1)/(M - 1)$, cu $0 \leq lin \leq N - 1$, și $0 \leq col \leq M - 1$ (relația 1)

Aceasta înseamnă că toate punctele de întâlnire mențin un raport constant între indicele liniei și indicele coloanei, egal cu $(N - 1)/(M - 1)$.

Prin urmare, numărul punctelor de întâlnire este legat de $GCD(N - 1, M - 1)$ și coordonatele acestor puncte se pot calcula direct din relația 1.

6.9 Cod-sursă pentru problema Special

```cpp
#include <bits/stdc++.h>
using namespace std;
ifstream  fin("special.in");
ofstream fout("special.out");
```

```
int c,n,m,k,x,nras,y,ri,rj[1000002],nrs11,nrs22;
int main()
{
    fin>>c>>n>>m>>k;
    if(c==1)
    {
        rj[0]=1;
        for(int j=1; j<=m-1; j++)
        {
            y=(4*j)%k;
            rj[y]++;
        }
        for(int i=0; i<=n-1; i++)
        {
            ri=(15*i+2025)%k;
            if(ri<=11)nrs11+=rj[11-ri];
            else nrs11+=rj[k-(ri-11)];
            if(ri<=22)nrs22+=rj[22-ri];
            else nrs22+=rj[k-(ri-22)];
        }
        fout<<nrs11+nrs22;
    }
    if(c==2)
    {
        int G=__gcd(n-1,m-1);
        int paslin=(n-1)/G;
        int pascol=(m-1)/G;
        for (int kk=0; kk<=G; kk++)
        {
            int lin=kk*paslin;
            int col=kk*pascol;
            x=(15*lin+4*col+2025)%k;
            if (x>100)
            {
                int nr1=0,nr2=0;
                while (x)
                {
                    if (x%10==1) nr1++;
                    if (x%10==2) nr2++;
                    x/=10;
                }
                if (nr1>=2 || nr2>=2) nras++;
            }
        }
        fout<<nras<<'\n';
    }
    return 0;
}
```

Capitolul 7

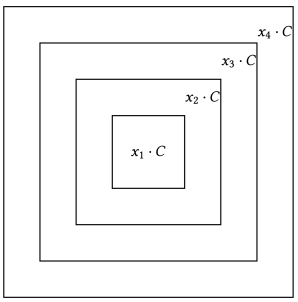
ONI 2025, clasa a VII-a

7.1 Problema Alvn

Propusă de: stud. Marcu Mihai, Delft University of Technology

Veveriţele ALVN şi prietenii săi Simon şi Theodore au fost afectaţi de noua criză de ghinde, aşa că au plecat de acasă în căutarea hranei. Din fericire, după o perioadă de căutări, au descoperit o grădină cu N rânduri de stejari cu câte M stejari pe fiecare rând. Fiecare stejar are alocată câte o parcelă de formă pătrată şi de dimensiuni identice. Fiecare stejar este bătrân şi are ramuri mari, astfel încât produce ghinde care pot cădea nu doar în parcela în care se află, ci şi în parcelele adiacente. Fiecare stejar are un coeficient de producţie al ghindelor C şi va produce ghinde conform următoarei distribuţii:

- Produce un număr $x_1 \cdot C$ de ghinde în parcela proprie (centru).
- Un număr $x_2 \cdot C$ de ghinde ajung în parcelele din inelul imediat exterior (parcelele adiacente direct), ca în desenul alăturat.
- Un număr $x_3 \cdot C$ de ghinde ajung în celulele din al doilea inel şi aşa mai departe, pentru fiecare inel exterior.



Acest model continuă până la cel mai îndepărtat inel. Fiecare stejar are cel mult k inele, incluzând parcela în care se află, iar şirul $x_1, x_2, \ldots x_k$ este ordonat descrescător astfel încât stejarul produce cele mai multe ghinde în parcela în care se află, iar numărul scade treptat în parcelele din inelele mai îndepărtate.

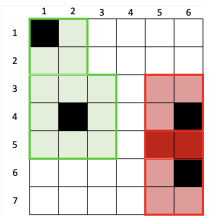
Inelele sunt pătratice şi concentrice, putând fi incomplete, în funcţie de poziţia stejarului în grădină.

Dacă în grădină este doar grupul format din ALVN şi prietenii săi, aleg pentru grup parcela cu numărul maxim de ghinde.

Dacă în grădină sunt două grupuri de veveriţe, acestea decid, pentru a nu exista supărări, ca ambele grupuri să aleagă propriile parcele, respectând următoarele reguli:

1. Pot mânca doar din copaci ale căror inele nu au nicio parcelă în comun.
2. Vor încerca să maximizeze numărul total de ghinde pe care le consumă.

De exemplu, în imaginea alăturată, dacă sunt două grupuri de veveriţe, ele se pot aşeza pe parcelele (1,1) şi (4,2), întrucât inelele copacilor (reprezentate cu verde) doar se ating, nu au parcele comune. În schimb, veveriţele nu se pot aşeza în parcelele (4,6) şi (6,6), deoarece inelele au în comun parcelele din poziţiile (5,5) şi (5,6) (în imagine sunt reprezentate cu roşu, inclusiv cele comune).



Se cunosc N, M, coeficienții fiecărui stejar din grădină, k, și valorile $x_1, x_2, \ldots x_k$, cu semnificația din enunț.

Cerințe

1. Determinați S, numărul maxim de ghinde pe care le poate consuma grupul lui ALVN, când ei sunt singuri în grădină.
2. Determinați T, numărul total de ghinde consumate de două grupuri de veverițe aflate în grădină.

Date de intrare

Pe prima linie a fișierului `alvn.in` se află un număr natural p, care reprezintă cerința (1 sau 2). Pe a doua linie se află două numere naturale N și M, cu semnificația din enunț. Următoarele N linii conțin câte M valori, reprezentând coeficienții de ghinde produse de fiecare stejar, în ordine, rând după rând și pentru fiecare rând, în ordinea parcelelor pe care se află. Pe linia $N+3$ se află valoarea k, cu semnificația din enunț. Pe linia următoare, se află k valori, reprezentând coeficienții $x_1, x_2, \ldots, x_k$, cu semnificația din enunț.

Date de ieșire

Fișierul de ieșire `alvn.out` va conține un singur număr natural, astfel:

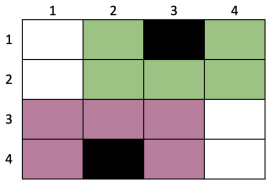
- Dacă $p = 1$, atunci se va afișa numărul S, determinat la cerința 1.
- Dacă $p = 2$, se va afișa numărul T, determinat la cerința 2.

Restricții

- $1 \leq N, M \leq 700$
- $1 \leq K \leq \min(N, M, 200)$
- $0 \leq x_k \leq 100$, pentru orice $k \in \{1, 2, \ldots, K\}$
- $0 \leq$ coeficientul fiecărui stejar ≤ 100
- Valorile $x_1, x_2, \ldots, x_k$ respectă relația $x_1 \geq x_2 \geq x_3 \geq \cdots \geq x_k$

#	Puncte	Restricții
1	10	$p = 1$ și $N, M \leq 100, K \leq 10$
2	35	$p = 1$ și nu există restricții suplimentare
3	20	$p = 2$ și $K \leq 10$
4	35	$p = 2$ și nu există restricții suplimentare

Exemple

alvn.in	alvn.out	Explicații
1 4 4 1 0 1 0 0 2 0 1 1 0 3 0 0 1 0 1 3 4 2 1	25	Numărul de ghinde din fiecare celulă este: 13 13 15 9 13 23 18 16 15 18 25 14 9 16 14 14 În parcela $(3,3)$ sunt ghinde din următorii stejari, din parcelele: $(3,3)$: $3 \cdot 4 = 12$, $(2,2)$: $2 \cdot 2 = 4$, $(2,4)$: $2 \cdot 1 = 2$, $(4,2)$: $2 \cdot 1 = 2$, $(4,4)$: $2 \cdot 1 = 2$, $(1,1)$: $1 \cdot 1 = 1$, $(1,3)$: $1 \cdot 1 = 1$, $(3,1)$: $1 \cdot 1 = 1$. Deci, în total sunt 25 de ghinde.
2 4 4 1 0 1 0 0 2 0 1 1 0 3 0 0 1 0 1 2 2 1	11	Veverițele se vor așeza în parcelele (1,3) și (4,2). Inelele acestor copaci nu se vor intersecta. Parcelele din inelele stejarului din $(1,3)$ sunt verzi, iar cele ale stejarului din $(4,2)$ sunt mov. Parcela $(1,3)$ va avea valoarea 5, iar parcela $(4,2)$ va avea valoarea 6. 

7.2 Rezolvarea problemei Alvn

Propusă de: prof. Adrian Panaete, Colegiul Național „August Treboniu Laurian", Botoșani

Cerința 1.

În rezolvare vom utiliza două matrice: SL, în care calculăm sumele parțiale pe linii, și SC, în care calculăm sumele parțiale pe coloane.

Astfel, $SL[i][j]$ va reține suma elementelor de pe linia i, elemente care se găsesc pe coloanele de la 1 la j, iar $SC[i][j]$ reține suma elementelor de pe coloana j, elemente care se găsesc de la linia 1 la linia i.

Putem observa că, pentru fiecare parcelă, se vor calcula, pentru fiecare poziție la distanța w de parcela curentă, $x[w] \cdot val$. Astfel, pentru toate valorile la distanța w de o anumită parcelă (de pe inelul w, pornind de la parcela noastră), acestea se vor înmulți cu $x[w]$. Vom încerca să calculăm mai rapid suma acestor parcele folosind matricele de sume parțiale definite anterior. Astfel:

- Suma de sus este $SL[i-w+1][j+w-1] - SL[i-w+1][j-w]$;
- Suma din dreapta este $SC[i+w-1][j+w-1] - SC[i-w+1][j+w-1]$;
- Suma de jos este $SL[i+w-1][j+w-2] - SL[i+w-1][j-w]$;
- Suma din stânga este $SC[i+w-2][j-w+1] - SC[i-w+1][j-w+1]$.

Observație: Aceste sume ar putea fi calculate cu modificarea indicilor, dacă aceștia nu se încadrează în intervalele de la 1 la N în cazul liniilor, respectiv de la 1 la M în cazul coloanelor. În

particular, sumele vor fi 0 dacă linia sau coloana la care ne referim nu este validă. Sumele se vor calcula pentru $w = \overline{1, K}$, iar complexitatea timp este $O(N \cdot M \cdot K)$.

Cerința 2.

O observație este că doi copaci nu au nicio parcelă în comun dacă diferența dintre liniile lor sau coloanele lor este mai mare sau egală cu $2 \cdot K - 1$.

Pentru soluția eficientă, vom avea nevoie de patru vectori: $maxUD$, $maxLR$, $maxDU$, $maxRL$.

- $maxUD[i]$ reprezintă elementul maxim care are linia mai mică sau egală cu i;
- $maxLR[i]$ reprezintă elementul maxim care are coloana mai mică sau egală cu i;
- $maxDU[i]$ reprezintă elementul maxim care are linia mai mare sau egală cu i;
- $maxRL[i]$ reprezintă elementul maxim care are coloana mai mare sau egală cu i.

Astfel, pentru fiecare element (i, j), parcela maximă al cărei stejar nu are alte parcele în comun cu stejarul de pe parcela (i, j) va fi maximul dintre:

- $maxUD[i - 2 \cdot K + 1]$
- $maxLR[j - 2 \cdot K + 1]$
- $maxDU[i + 2 \cdot K - 1]$
- $maxRL[j + 2 \cdot K - 1]$

Complexitatea în timp este $O(N \cdot M \cdot K)$, deoarece trebuie să calculăm matricea de la cerința 1.

7.3 Cod-sursă pentru problema Alvn

```
#include <iostream>
#include <fstream>
#include <assert.h>

using namespace std;
ifstream f("alvn.in");
ofstream g("alvn.out");
const int N=702;
/// pentru i,j,k fixate
/// U=i-k,D=i+l,L=j-k,R=j+k
int cer,n,m,k,SOL,M[N][N],SL[N][N],SC[N][N],sol[N][N],x[N];
int maxUD[N],maxDU[N],maxLR[N],maxRL[N];
int sumaLin(int lin,int L,int R)
{
    if(lin<1||lin>n)return 0;
    L=max(L,1);
    R=min(R,m);
    return SL[lin][R]-SL[lin][L-1];
}
int sumaCol(int col,int U,int D)
{
    if(col<1||col>m)return 0;
    U=max(U,1);
    D=min(D,n);
    return SC[D][col]-SC[U-1][col];
}
int main()
{
    f>>cer>>n>>m;
    for(int i=1; i<=n; i++)
        for(int j=1; j<=m; j++)
```

```
        {
            f>>M[i][j];
            SL[i][j]=SL[i][j-1]+M[i][j];
            SC[i][j]=SC[i-1][j]+M[i][j];
        }
    f>>k;
    for(int i=0; i<k; i++)
        f>>x[i];
    for(int i=1; i<=n; i++)
        for(int j=1; j<=m; j++)
            sol[i][j]=x[0]*M[i][j];
    for(int K=1; K<k; K++)
        for(int i=1,U=i-K,D=i+K; i<=n; i++,U++,D++)
            for(int j=1,L=j-K,R=j+K; j<=m; j++,L++,R++)
                sol[i][j]+=x[K]*(sumaLin(U,L,R-1)+sumaCol(R,U,D-1)+
                                 sumaLin(D,L+1,R)+sumaCol(L,U+1,D));
    /// determinam maximul pe fiecare linie
    for(int i=1; i<=n; i++)
        for(int j=1; j<=m; j++)
        {
            maxLR[j]=maxRL[j]=max(maxLR[j],sol[i][j]);/// maxime pe coloane
            maxUD[i]=maxDU[i]=max(maxUD[i],sol[i][j]);/// maxime pe linii
            SOL=max(SOL,sol[i][j]);
        }
    if(cer==1)
    {
        g<<SOL<<'\n';
        return 0;
    }
    SOL=0;
    for(int i=2; i<=n; i++)maxUD[i]=max(maxUD[i],maxUD[i-1]);
    for(int j=2; j<=m; j++)maxLR[j]=max(maxLR[j],maxLR[j-1]);
    for(int i=n-1; i>=1; i--)maxDU[i]=max(maxDU[i],maxDU[i+1]);
    for(int j=m-1; j>=1; j--)maxRL[j]=max(maxRL[j],maxRL[j+1]);
    for(int i=1,U=i-2*k+1,D=i+2*k-1; i<=n; i++,U++,D++)
        for(int j=1,L=j-2*k+1,R=j+2*k-1; j<=m; j++,L++,R++)
        {
            int other=0;
            if(U>=1)other=max(other,maxUD[U]);
            if(L>=1)other=max(other,maxLR[L]);
            if(D<=n)other=max(other,maxDU[D]);
            if(R<=m)other=max(other,maxRL[R]);
            if(other)
                SOL=max(SOL,sol[i][j]+other);
        }
    g<<SOL;
    return 0;
}
```

7.4 Problema Conturi

Propusă de: prof. Veronica-Raluca Costineanu, Colegiul Național „Ștefan cel Mare", Suceava

Alina, managerul unui lanț de magazine, este responsabilă de gestiunea tranzacțiilor bancare din cadrul acestora. Ea lucrează cu conturi bancare și cunoaște sumele de bani (soldul) existente în fiecare dintre acestea. După ce a ales banca cu care va colabora, stabilește următoarele reguli:

- tranzacțiile trebuie efectuate în ordinea în care apar;
- trebuie să deschidă cât mai puține conturi;
- fiecărui cont nou deschis i se asociază un cod unic egal cu numărul conturilor care au fost deschise de Alina până atunci;
- la o tranzacție de **depunere** suma trebuie depusă în întregime într-un singur cont; contul se alege astfel încât această sumă să fie strict mai mare decât ultima sumă depusă în acest cont. Dacă există mai multe astfel de conturi se alege cel pentru care ultima sumă depusă este maximă, iar dacă există mai multe astfel de conturi se alege cel care are codul asociat minim. Dacă în niciunul dintre conturile existente nu poate fi depusă suma conform precizărilor, se deschide un nou cont și se depune suma în acesta.
- la o tranzacție de **retragere** suma necesară se poate obține din unul sau mai multe conturi. Se alege pentru retragere contul care are soldul cel mai mic, iar dacă aceasta este mai mare decât suma necesară se actualizează soldul (prin scăderea din acesta a sumei necesare), iar procesul se încheie. Dacă soldul este insuficient, se actualizează suma necesară (prin scăderea din aceasta a soldului), iar contul respectiv se închide și procesul se reia cu alegerea unui alt cont adecvat. Dacă există mai multe conturi cu același sold minim se alege pentru retragere cel în care ultima sumă depusă este maximă; dacă sunt mai multe astfel de conturi se alege cel care are codul asociat maxim.

Se cunosc N, numărul tranzacțiilor și N numere întregi nenule a_1, a_2, ..., a_N, reprezentând, în această ordine, sumele de tranzacționat (un număr pozitiv indică o sumă care urmează a fi depusă, iar un număr negativ reprezintă o sumă care urmează a fi retrasă).

Cerințe

După procesarea celor N tranzacții, ajutați-o pe Alina să determine:

1. numărul de conturi rămase active.
2. soldul maxim care se găsește într-un cont dintre cele rămase active.

Date de intrare

Fișierul de intrare `conturi.in` conține pe prima linie două numere naturale, C și N, unde C este numărul cerinței care trebuie rezolvată (care poate fi doar 1 sau 2), iar N are semnificația din enunț. Pe a doua linie, separate prin câte un spațiu se află cele N valori întregi nenule a_1, a_2, ..., a_N cu semnificația din enunț.

Date de ieșire

Fișierul de ieșire `conturi.out` conține numărul determinat pentru cerința C.

Restricții

- $1 \leq n \leq 10^5$;
- $-10^{18} \leq a_i \leq 10^{18}$, pentru oricare $1 \leq i \leq N$;

- $a_i \neq 0$, pentru oricare $1 \leq i \leq N$;
- sumele care trebuie retrase nu depășesc totalul soldurilor din conturile active la momentul retragerii;
- sunt cel mult 10 retrageri.

#	Puncte	Restricții
1	39	$C = 1$
2	61	$C = 2$

Exemple

conturi.in	conturi.out	Explicații
1 5 10 15 5 10 20	2	În contul cu codul 1 se depun sumele: 10, 15, 20, deci acesta are soldul 10 + 15 + 20 = 45, iar în contul cu codul 2 se depun sumele 5, 10, deci acesta are soldul 5 + 10 = 15. Sunt 2 conturi active.
2 5 10 15 5 10 20	45	Tranzacțiile sunt conform descrierii de mai sus, iar soldul maxim este 45, în contul 1.
1 7 10 15 5 10 20 -15 35	1	Până în momentul tranzacției de retragere, se procedează ca la exemplul 1. Suma 15 se retrage din contul al doilea care astfel se închide (15-15=0), iar în singurul cont rămas activ se depune suma 35.
2 7 10 15 5 10 20 -15 35	80	Se procedează ca la exemplul 3 deci în contul activ soldul devine 45+35=80.
2 7 10 15 5 10 20 -45 35	50	Până în momentul tranzacției de retragere, se procedează ca la exemplul 1. Pentru a retrage suma 45 se efectuează o retragere din contul 2 (cu soldul 15) care se închide; suma de retras actualizată (45-15=30) este extrasă în continuare din contul 1 (cu soldul 45), iar după depunerea următoare (de 35) soldul final al acestuia va fi 50.

7.5 Rezolvarea problemei Conturi

Pentru a reține organizat informațiile despre conturile existente, putem folosi o structură `cont`, cu atributele:

- `id`: număr natural, unic pentru fiecare cont;
- `sold`: șir de cifre, reprezentând suma totală depusă în cont;
- `ultimAdăugat`: număr natural, reprezentând ultima sumă depusă în contul respectiv;

Procesând tranzacțiile pe rând, în ordinea în care apar, avem două cazuri: dacă valoarea citită *val* este pozitivă, vom avea un caz de depunere, iar dacă este negativă, vom avea un caz de retragere.

Putem reține conturile ordonate descrescător după ultima sumă adăugată, iar în caz de egalitate, crescător după `id`. În acest fel, în cazul unei depuneri, vom căuta binar poziția corespunzătoare la

care trebuie să facem depunerea (contul cu ultima valoare adăugată maximă, dar strict mai mică decât val), iar dacă poziția corespunzătoare este în afara intervalului de indici în care reținem conturile active, vom crea un cont nou.

Pentru operația de retragere, vom reordona conturile descrescător după sold, iar în caz de egalitate, după ultima valoarea adăugată și, în final, crescător după `id`. Putem astfel să realizăm retragerea începând de la contul cu cel mai mic sold (dreapta), spre contul cu cel mai mare sold (stânga). La fiecare pas, vom compara soldul contului curent $cont[i]$ cu val, iar dacă nu există sold suficient, $val = val - cont[i]$ și contul curent dispare. Ne deplasăm atât timp cât $|val| > 0$. După finalizarea extragerii, vom reordona conturile descrescător după ultima valoare adăugată.

În final, vom afișa răspunsul corespunzător cerinței p. Complexitatea timp a soluției este $O(n \cdot k \cdot \log n)$, unde k reprezintă numărul de ștergeri.

7.6 Cod-sursă pentru problema Conturi

```
#include <fstream>
#include <algorithm>
using namespace std;

#define N 100005
#define nrCif 30

ifstream fin ("conturi.in");
ofstream fout("conturi.out");

struct contBancar
{
    int cod;
    short sum[nrCif];
    long long ultima;
} v[N];
int n, C, k, nrCont;

inline int cmp(short a[], short b[])
{
    if(a[0] > b[0]) return 1;
    if(a[0] < b[0]) return -1;
    for(int i = a[0]; i >= 1; --i)
        if(a[i] > b[i]) return 1;
        else if(a[i] < b[i]) return -1;
    return 0;
}

inline void aduna(short a[], long long d)
{
    int i;
    for(i = 1; i <= a[0]; ++i)
        d += a[i], a[i] = d % 10, d /= 10;
    while(d)
        a[++a[0]] = d % 10, d/= 10;
}
inline void scade(short a[], short b[])
{
    int i, j;
    for(i = b[0] + 1; i <= a[0]; ++i)
        b[i] = 0;
    for (i = 1; i <= a[0]; i++)
        if(a[i] >= b[i])
```

```
            a[i] -= b[i];
        else
        {
            j=i+1;
            while(a[j] == 0 && j < a[0])
                a[j++] = 9;
            a[j]--;
            a[i] = 10 + a[i] - b[i];
        }
    while(a[0] > 1 && a[a[0]] == 0) a[0]--;
}

void afis(short a[])
{
    for(int i = a[0]; i >= 1; --i)
        fout << a[i];
    fout << '\n';
}
inline bool cmpSum(contBancar A, contBancar B)
{
    int c = cmp(A.sum, B.sum);
    return c > 0 || (c == 0 && A.ultima < B.ultima) ||
           (c == 0 && A.ultima == B.ultima && A.cod < B.cod);
}
inline bool cmpUltima(contBancar A, contBancar B)
{
    return A.ultima > B.ultima || (A.ultima == B.ultima && A.cod < B.cod);
}
inline int cautaPozitia(long long x)
{
    int st = 1, dr = k, m;
    while(st <= dr)
    {
        m = (st + dr) / 2;
        if(v[m].ultima >= x) st = m + 1;
        else dr = m - 1;
    }
    return st;
}
int main()
{
    long long x;
    int p;
    fin >> C >> n;
    for(int i = 1; i <= n; ++i)
    {
        fin >> x;
        if(x > 0)
        {
            p = cautaPozitia(x);
            if (p > k)
               v[++k].ultima=x, v[k].sum[0]=0, aduna(v[k].sum, x), v[k].cod=++nrCont;
            else v[p].ultima = x, aduna(v[p].sum, x);
        }
        else
        {
            x *= -1;
            short s[nrCif] = {};
            while(x)
                s[++s[0]] = x % 10, x /= 10;
            sort(v + 1, v + k + 1, cmpSum);
```

```
            while(s[0] > 0 && k > 0)
            {
                if(cmp(s, v[k].sum) > 0)
                {
                    scade(s, v[k].sum);
                    v[k].sum[0] = v[k].ultima = v[k].cod = 0;
                    --k;
                }
                else if(cmp(s, v[k].sum) == 0)
                {
                    v[k].sum[0] = v[k].ultima = v[k].cod = 0;
                    --k;
                    s[0] = 0;
                }
                else
                {
                    scade(v[k].sum, s);
                    s[0] = 0;
                }
            }
            sort(v + 1, v + k + 1, cmpUltima);
        }
    }
    if(C == 1) fout << k << '\n';
    else
    {
        sort(v + 1, v + k + 1, cmpSum);
        afis(v[1].sum);
    }
    return 0;
}
```

7.7 Problema Succesori

Propusă de: prof. Adrian Panaete, Colegiul Național „August Treboniu Laurian", Botoșani

Rareș a învățat la ora de informatică despre conceptul de succesor. **Succesorul** unui număr natural nenul X este numărul $S(X)$ obținut din X, astfel: fiecare cifră strict mai mică decât 9 se înlocuiește cu cifra mai mare cu o unitate, iar cifra 9 se înlocuiește cu cifra 0. Din numărul obținut, se elimină cifrele nule aflate pe primele poziții, iar numărul 0 nu are succesor. De exemplu:

- $S(12) = 23$;
- $S(944) = 55$;
- $S(795) = 806$;
- $S(999) = 0$;
- $S(999912) = 23$;
- $S(9) = 0$.

Mihai, colegul lui Rareș, îi propune acestuia următoarea problemă: se dau două numere naturale n, p și un șir de n numere naturale $x_1, x_2, \ldots, x_n$. Se scriu cele n numere dispuse unul sub celălalt, câte unul pe un rând. Pe fiecare rând, se adaugă numere, după următoarea regulă:

- Dacă ultimul număr de pe un rând este nenul, atunci, la rândul respectiv, se adaugă succesorul acelui ultim număr.
- După ce s-au completat toate rândurile, și astfel fiecare rând se termină cu valoarea 0, se iau toate numerele de pe cele n rânduri și se formează un șir.
- Se ordonează crescător șirul obținut.

De exemplu, dacă $n = 3$, iar cele 3 numere date inițial de Mihai sunt $78, 9552$ și 752, avem inițial 3 rânduri, fiecare cu câte un număr:

1.	78
2.	9552
3.	752

După ce se completează rândurile cu succesori, cele 3 rânduri complete vor fi:

1.	78 89 90 1 2 3 4 5 6 7 8 9 0
2.	9552 663 774 885 996 7 8 9 0
3.	752 863 974 85 96 7 8 9 0

Cu toate numerele, se formează șirul crescător următor:

0 0 0 1 2 3 4 5 6 7 7 7 8 8 8 9 9 9 78 85 89 90 96 663 752 774 863 885 974 996 9552

Cerințe

Cunoscând numerele naturale n, p și cele n numere naturale inițiale, determinați numărul situat pe poziția p în șirul final obținut, dacă pozițiile sunt numerotate începând cu 1.

Date de intrare

Pe prima linie a fișierului `succesori.in` se află două numere naturale n și p, separate prin spațiu, cu semnificația din enunț. Pe următoarele n linii ale fișierului, se află cele n numere naturale nenule $x_1, x_2, \ldots, x_n$, câte unul pe linie, cu semnificația din enunț.

Date de ieșire

Pe singura linie a fișierului `succesori.out` se va afla un singur număr natural, reprezentând numărul situat pe poziția p în șirul sortat, format după regula din enunț.

Restricții

- $1 \leq n \leq 10^6$;
- $1 \leq x_n < 10^{18}$;
- $p <$ numărul total de numere din șirul ordonat obținut.

#	Puncte	Restricții
1	7	$n \leq 1000$
2	15	$1000 < n \leq 10^5$
3	26	$10^5 < n \leq 5 \cdot 10^5$
4	52	$5 \cdot 10^5 < n \leq 10^6$

Exemple

succesori.in	succesori.out	Explicații
3 23 78 9552 752	96	Exemplul este explicat in enunțul problemei.
5 65 23314 12 10019324124 5566778 1423	89	Se scriu pe rând succesorii pe fiecare rând, conform regulii. În șirul sortat obținut, pe poziția 65, se află numărul 89.
10 15 145123187 292253412 545220314 645144712 945114524 245100187 525656715 443123187 577856712 845223886	2	În șirul sortat obținut după scrierea succesorilor, pe poziția 15, se află numărul 2.
10 301 145123187698456712 122133567292253412 545220314744452740 645123187698456712 945123187222400714 245100187698456712 245123180025656715 443123187698456716 545223112577856712 745223886198456718	34596	În șirul sortat obținut după scrierea succesorilor, pe poziția 301, se află numărul 34596.

7.8 Rezolvarea problemei Succesori

O posibilă soluție generează toți succesorii elementelor citite, ordonează crescător șirul format de aceștia și afișează valoarea aferentă. Această soluție este ineficientă, datorită restricțiilor de numere mari.

Având în vedere numărul mare de succesori care vor fi generați, o soluție eficientă nu va memora toți succesorii fiecărui număr. Astfel, vom determina, prima dată, numărul de cifre și cifra dominantă a rezultatului. În acest scop, putem face următoarele observații:

- Pentru fiecare număr de cifre și fiecare cifră dominantă, o valoare poate genera cel mult un succesor;
- Dacă vom genera toți succesorii unui număr, fără a-i memora, putem număra pentru fiecare număr de cifre și pentru fiecare cifră dominantă câți succesori generează toate valorile inițiale;
- Pentru un anumit număr de cifre C, cifra dominantă a primului succesor generat având C cifre nu depinde de valoarea întregului număr, ci doar de cifrele de la pozițiile C, respectiv $C+1$. Mai precis, dacă inițial, cele două cifre sunt x și y, atunci, când y ajunge la valoarea 0, x va ajunge la valoarea $t = (y + 10 - x)\%10$ și t va fi cifra dominantă a primului succesor generat cu C cifre. În consecință, vor fi generați $10 - t$ succesori cu C cifre, iar aceste cifre vor fi $t, t + 1, \ldots, 9$.

Notăm cu $sol[C][d]$ numărul total de succesori cu C cifre care au cifra dominantă d.

Folosind ultima observație, putem contoriza pentru fiecare valoare și fiecare număr de cifre o unitate doar pentru $sol[C][t]$. Apoi, sumând parțial de la 1 la 9 pe cifre, vom obține pentru fiecare cifră numărul real de succesori care vor fi generați. Această abordare reduce foarte mult timpul de calculare al valorilor $sol[C][d]$.

Numărul de cifre și cifra dominantă a rezultatului pot fi acum determinate parcurgând matricea *sol* în ordinea crescătoare a numărului de cifre și, apoi, în ordinea crescătoare a cifrelor dominante. Sumând toate aceste valori, până în momentul în care suma devine mai mare sau egală decât poziția dorită, determinăm exact cifra dominantă și numărul de cifre al rezultatului. Deoarece matricea *sol* are cel mult 18 linii și 9 coloane, numărul de adunări va fi cel mult 162.

În continuare, bazându-ne pe prima observație și folosind aceeași tehnică, putem genera, pentru fiecare valoare inițială, cel mult un succesor cu numărul de cifre dorit și cifra dominantă dorită. Scăzând din poziția cerută suma numărului de succesori, având strict mai puține cifre sau același număr de cifre, dar cu cifra dominantă strict mai mică, putem determina pe ce poziție, printre succesorii generați, vom găsi rezultatul. Sortând acești succesori, determinăm și care este valoarea cerută. Complexitatea finală a algoritmului este $O(n \cdot maxC + n \log n)$, unde $maxC$ este numărul maxim de cifre pentru valorile din șirul inițial.

De menționat că, pentru a determina al k-lea cel mai mic termen dintr-un șir, se poate folosi algoritmul de partiționare folosit la `quick sort` sau funcția `std::nth_element`, însa nu este necesar pentru obținerea punctajului maxim.

7.9 Cod-sursă pentru problema Succesori

```
#include <fstream>
#include <iostream>
#include <algorithm>
using namespace std;
ifstream f("succesori.in");
ofstream g("succesori.out");
typedef int64_t Int;
const int N=1000010;
int n,m,k,poz,lo,hi,c,d,cif[30][30],sum[30][30],su[30][30];
Int v[N],V[N],P;

void precalc(Int X)
```

```
{
    m++;
    /// se considera ultimele doua cifre pentru numarul X
    int u=X%10;
    X/=10;
    int p=X%10;
    X/=10;
    int c=1;
    /// cat timp am cifre in X sau am cifra u sau am cifa p
    while(p||u||X)
    {
        if(p!=u)
        {
            int cMin=cif[u][p];
            sum[c][cMin]++;
            m+=10-cMin;
        }
        u=p;
        p=X%10;
        X/=10;
        c++;
    }
}

int main()
{
    f>>n>>poz;
    // cif[u][p] = care va fi prima cifra daca cifra de pe o pozitie este u
    //                         si este precedata de cifra p
    // raspuns cand cifra p va deveni 0 cifra u va deveni cif[u][p]=(10-p+u)%10;
    for(int u=0; u<10; u++)
        for(int p=0; p<10; p++)
            cif[u][p]=(10-p+u)%10;
    for(int i=1; i<=n; i++)
    {
        f>>v[i];
        precalc(v[i]);
    }
    for(int c=1; c<=18; c++)
        for(int d=1; d<=9; d++)
            sum[c][d]+=sum[c][d-1];
    lo=1;
    hi=n;
    while(1)
    {
        if(lo<=poz&&poz<=hi)
            break;
        d++;
        if(d==10)
        {
            d=1;
            c++;
        }
        if(sum[c][d])
        {
            lo=hi+1;
            hi+=sum[c][d];
        }
    }
    poz=poz-lo+1;
    P=1;
```

```
    for(int i=1; i<c; i++)P=P*10;
    for(int i=1; i<=n; i++)
    {
        Int X=v[i],Z=0;
        int u=X/P%10,p=X/P/10%10;
        ///cout<<u<<' '<<p<<'\n';
        int ord=cif[u][p];
        ///cout<<ord<<'\n';
        if(ord>0&&ord<=d)
        {
            ord=cif[d][u];
            Int PP=1;
            for(int j=1; j<=c; j++,PP*=10,X/=10)
                Z+=PP*((X%10+ord)%10);
            V[++k]=Z;
            ///cout<<k<<' '<<V[k]<<'\n';
        }
    }
    sort(V+1,V+k+1);
    g<<V[poz];
    return 0;
}
```

Capitolul 8

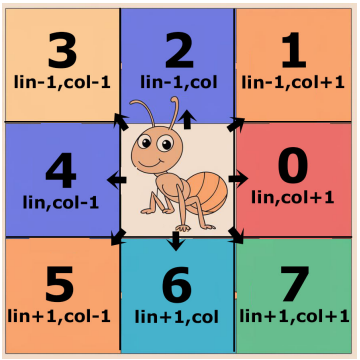
ONI 2025, clasa a VIII-a

8.1 Problema Mușuroi

Propusă de: stud. Andrei Boacă, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iași

Un mușuroi format din mai multe celule este reprezentat ca o matrice cu N linii, numerotate de la 1 la N, și M coloane, numerotate de la 1 la M, fiecare element al matricii corespunzând unei celule. Poziția unei celule din mușuroi este identificată prin linia și coloana pe care se află. În fiecare celulă a mușuroiului este desenată o săgeată, care indică sensul în care o furnică aflată în poziția respectivă se va deplasa. Săgețile sunt codificate cu numere de la 0 la 7. Dacă furnica se află în celula (lin, col) la momentul t, atunci la momentul $t + 1$ ea va ajunge în poziția:

- $(lin, col + 1)$, dacă săgeata este 0;
- $(lin - 1, col + 1)$, dacă săgeata este 1;
- $(lin - 1, col)$, dacă săgeata este 2;
- $(lin - 1, col - 1)$, dacă săgeata este 3;
- $(lin, col - 1)$, dacă săgeata este 4;
- $(lin + 1, col - 1)$, dacă săgeata este 5;
- $(lin + 1, col)$, dacă săgeata este 6;
- $(lin + 1, col + 1)$, dacă săgeata este 7.



În figura alăturată sunt ilustrate săgețile, fiind indicată poziția în care ajunge furnica pentru fiecare dintre ele.

Cerințe

Cunoscând reprezentarea mușuroiului, scrieți un program care să răspundă la Q întrebări de forma lin_A col_A lin_B col_B K, cu semnificația "Dacă o furnică pornește de la celula (lin_A, col_A) la momentul de timp $t = 0$, la ce moment de timp va ajunge a K-a oară în poziția (lin_B, col_B)?". Se garantează că furnica poate trece de cel puțin K ori prin celula (lin_B, col_B) dacă pornește din celula (lin_A, col_A).

Date de intrare

Fișierul de intrare `musuroi.in` conține pe prima linie numerele naturale N și M cu semnificația din enunț. Pe următoarele N linii se află câte M numere cuprinse între 0 și 7, ce reprezintă săgețile din fiecare celulă a matricii. Pe linia $N + 2$ se află numărul natural Q, reprezentând

numărul de întrebări, iar pe următoarele Q linii se află cele Q întrebări, câte o întrebare pe o linie, în forma descrisă mai sus. Valorile scrise pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `musuroi.out` conține Q linii, pe linia i aflându-se un număr ce reprezintă răspunsul pentru cea de a i-a întrebare din fișierul de intrare ($1 \leq i \leq Q$).

Restricții

- $2 \leq N, M \leq 1\,000$
- $1 \leq Q \leq 100\,000$
- În întrebările pentru care $K = 1$ pozițiile (lin_A, col_A) și (lin_B, col_B) pot să coincidă, caz în care răspunsul este 0.
- $1 \leq K \leq 1\,000\,000\,000$
- Se garantează că furnica, deplasându-se în sensul săgeților, va rămâne în mușuroi.

#	Puncte	Restricții
1	11	$K = 1, \quad 2 \leq N, M, Q \leq 100$
2	37	$K = 1$, fără restricții suplimentare
3	9	$1 < K \leq 7, \quad 2 \leq N, M, Q \leq 100$
4	43	$K > 1$, fără restricții suplimentare

Exemple

musuroi.in	musuroi.out
6 6 7 6 4 0 0 6 0 5 2 0 3 4 2 5 0 5 2 6 0 7 5 0 6 3 6 1 7 2 0 6 0 0 0 0 2 4 2 2 4 2 6 1 4 2 6 6 1	5 6
6 6 7 6 4 0 0 6 0 5 2 0 3 4 2 5 0 5 2 6 0 7 5 0 6 3 6 1 7 2 0 6 0 0 0 0 2 4 2 2 4 2 6 3 4 2 6 6 5	15 22

Explicație

Exemplul 1. Prima întrebare: furnica pleacă la momentul $t = 0$ din poziția $(2, 4)$. Traseul furnicii este: $(2, 4) \to (2, 5) \to (1, 4) \to (1, 5) \to (1, 6) \to (2, 6)$. La momentul $t = 5$ a ajuns la destinație, în poziția $(2, 6)$ pentru prima dată ($K = 1$). A doua întrebare: furnica pleacă la

momentul $t = 0$ din poziția $(4, 2)$. Traseul furnicii este: $(4, 2) \to (5, 3) \to (6, 4) \to (6, 5) \to (5, 5) \to (5, 6) \to (6, 6)$. La momentul $t = 6$ a ajuns la destinație, în poziția $(6, 6)$ pentru prima dată ($K = 1$).

Exemplul 2. Prima întrebare: furnica pleacă la momentul $t = 0$ din poziția $(2, 4)$. Traseul furnicii este: $(2, 4) \to (2, 5) \to (1, 4) \to (1, 5) \to (1, 6) \to (2, 6) \to (2, 5) \to (1, 4) \to (1, 5) \to (1, 6) \to (2, 6) \to (2, 5) \to (1, 4) \to (1, 5) \to (1, 6) \to (2, 6)$. La momentul $t = 15$ a ajuns la destinație, în poziția $(2, 6)$ pentru a treia oară ($K = 3$). A doua întrebare: furnica pleacă la momentul $t = 0$ din poziția $(4, 2)$. Traseul furnicii este: $(4, 2) \to (5, 3) \to (6, 4) \to (6, 5) \to (5, 5) \to (5, 6) \to (6, 6) \to (6, 5) \to (5, 5) \to (5, 6) \to (6, 6) \to (6, 5) \to (5, 5) \to (5, 6) \to (6, 6) \to (6, 5) \to (5, 5) \to (5, 6) \to (6, 6) \to (6, 5) \to (5, 5) \to (5, 6) \to (6, 6)$. La momentul $t = 22$ a ajuns la destinație, în poziția $(6, 6)$ pentru a cincea oară ($K = 5$).

8.2 Rezolvarea problemei Mușuroi

Soluție O(Q*N*M*K)

Se poate simula drumul pe care îl parcurge furnica, oprindu-ne în momentul în care am ajuns a K-a dată la celula de final. Această soluție se încadrează în timp pentru N, M, $Q \leq 100$ și $K \leq 5$.

Soluție O(Q*N*M)

Observația principală este că, având în vedere că furnica nu poate ieși din matrice, va putea parcurge o infinitate de pași prin matrice. Prin urmare, există o celulă prin care va trece de cel puțin două ori. Să notăm prima celulă pe care o va întâlni furnica a doua oară cu P_1. Este evident că, dacă pornind din celula $P1$ am putut ajunge încă o dată în celula $P1$ avem de fapt o secvență periodică de celule prin care trece furnica. Vom nota lungimea acestei secvențe periodice cu L. În consecință, orice drum al unei furnici care pornește din celula A va fi de forma A, A_2, A_3,..., P_1, P_2, P_3,... P_L, P_1, P_2, P_3, ... Pentru soluția $O(Q * N * M)$ putem simula drumul furnicii până când întâlnește a doua oară o celulă, moment în care știm că am găsit celula P_1. Astfel, celulele care au apărut între cele două apariții ale lui P_1 vor face parte din perioadă. Deci, pentru orice celulă din perioadă, dacă aceasta apare prima dată la momentul T, atunci va apărea următoarea dată la momentul $T + L$, a treia oară la momentul $T + 2 * L$, ș.a.m.d. Deci, putem calcula ușor timpul după care celula va apărea a K-a oară.

Soluție O(Q + N*M)

Pornind de la soluția anterioară, putem observa că nu este necesar ca de fiecare dată să parcurgem tot drumul până la celula P_1 pentru fiecare întrebare. Mai mult, putem precalcula celula P_1, distanța până la aceasta și lungimea perioadei pentru fiecare celulă din matrice. În plus, vom reține și distanța fiecărei celule ce face parte dintr-o perioadă către o celulă pe care o vom desemna celulă de start pentru acea perioadă (acea celulă poate fi aleasă arbitrar). Vom proceda în felul următor pentru a afla toate aceste informații.

Considerăm inițial toate celulele ca fiind nevizitate. Iterăm prin celule într-o ordine arbitrară, iar dacă aceasta nu a fost vizitată, vom începe să parcurgem drumul pornind din ea. Dacă la un moment dat înâlnim pe drum o celulă vizitată într-o iterație anterioară, atunci știm că pentru acea celulă am calculat deja informațiile necesare, prin urmare putem distribui acele informații corespunzător și către celulele vizitate în iterația curentă. Altfel, dacă întâlnim o celulă deja vizitată în iterația curentă, înseamnă că am găsit celula P_1 și lungimea perioadei pentru toate celulele parcurse în iterația curentă, deci le putem din nou actualiza corespunzător. Acum, putem

răspunde la fiecare întrebare în O(1), deoarece știm lungimea prefixului până la celula P_1, precum și lungimea L.

8.3 Cod-sursă pentru problema Mușuroi

```
#include <bits/stdc++.h>

using namespace std;
ifstream fin("musuroi.in");
ofstream fout("musuroi.out");
int diri[8]= {0,-1,-1,-1,0,1,1,1};
// deplasarea pe linii in functie de indicele sagetii
int dirj[8]= {1,1,0,-1,-1,-1,0,1};
// deplasarea pe coloane in functie de indicele sagetii
int n,m,v[1005][1005],cerinta,q;
int comp[1005][1005];
// comp[x][y] -> indicele secventei periodice in care va ajunge celula (x,y)
int dist[1005][1005],who[1005][1005];
bool oncycle[1005][1005];
int lg[1005*1005];
// lungimea fiecarei secvente periodice
void move(int &x,int &y)
// muta furnica de la celula (x,y) in directia sagetii
{
    int dx=diri[v[x][y]];
    int dy=dirj[v[x][y]];
    x+=dx;
    y+=dy;
}
int main()
{
    fin>>n>>m;
    for(int i=1; i<=n; i++)
        for(int j=1; j<=m; j++)
            fin>>v[i][j];
    int nrcomp=0;
    for (int px=1; px<=n; px++)
        for (int py=1; py<=m; py++)
            if (comp[px][py]==0)
            //celula nu a fost parcursa pana acum, deci incepem o parcurgere din ea
            {
              int x=px;
              int y=py;
              while (comp[x][y]==0)
              //parcurgem cat timp celula in care suntem nu a fost deja vizitata
              {
                comp[x][y]=-1;
                //marcam pentru a sti ca am trecut prin celula la iteratia curenta
                move(x,y);
              }
              if (comp[x][y]==-1)
              //celula in care ne oprim a fost vizitata in iteratia curenta
              //=> am gasit celula P
              {
                nrcomp++; // am gasit o noua secventa periodica
                int X=x;
                int Y=y;
                x=px;
                y=py;
                int cnt=0;
```

```
//lungimea secventei dinainte de a intra in secventa periodica
while (x!=X||y!=Y)
//gasim portiunea de secventa dinaintea celulei P
{   cnt++;
    move(x,y);
}
x=px; y=py;
while (x!=X || y!=Y)
//setam distantele de la celulele din afara secventei periodice
//pana la celula P
{
  comp[x][y]=nrcomp;
  dist[x][y]=cnt;
  who[x][y]=0;
  cnt--;
  move(x,y);
}
x=X;
y=Y;
cnt=0;
while (true) //gasim secventa periodica
{
   comp[x][y]=nrcomp;
   dist[x][y]=0;
   who[x][y]=cnt;
  //numarul de ordine al celulei in cadrul secventei periodice
   cnt++;
   oncycle[x][y]=1; //marcam ca este pe secventa periodica
   move(x,y);
   if (x==X&&y==Y)
      break;
}
lg[nrcomp]=cnt;
}
else
//celula in care ne-am oprit a fost vizitata in alta iteratie
//=> nu avem secventa periodica noua
{
    int X=x;
    int Y=y;
    int c=comp[X][Y];
    int cnt=dist[X][Y];
    //lungimea secventei dinainte de a intra in secventa periodica
    x=px;
    y=py;
    while (x!=X||y!=Y)
    //parcurgem bucata noua din secventa periodica
    {
        cnt++;
        move(x,y);
    }
    x=px;
    y=py;
    while(x!=X||y!=Y)
    {
        comp[x][y]=c;
        dist[x][y]=cnt;
        cnt--;
        who[x][y]=who[X][Y];
        move(x,y);
    }
```

```
                }
            }
    fin>>q;
    while (q--)
    {
      long long xa,ya,xb,yb,k;
      fin>>xa>>ya>>xb>>yb>>k;
      if (k==1)
         {
            if (!oncycle[xb][yb])
                // celula (xb,yb) se afla pe secventa periodica
                fout<<dist[xa][ya]-dist[xb][yb]<<'\n';
                // diferenta de distante pana la P
            else
            {
              int rez=dist[xa][ya]+
                      (who[xb][yb]-who[xa][ya]+lg[comp[xa][ya]])%lg[comp[xa][ya]];
             // distanta pana la P + distanta de la P la celula dorita
              fout<<rez<<'\n';
            }
         }
      else
        {
         long long rez=dist[xa][ya]+
                      (who[xb][yb]-who[xa][ya]+lg[comp[xa][ya]])%lg[comp[xa][ya]];
         //distanta pana la P + distanta de la P la celula dorita
         rez+=(k-1)*(1LL*lg[comp[xa][ya]]);
         //parcurgem de inca (k-1) ori secventa periodica
         fout<<rez<<'\n';
        }
    }
}
```

8.4 Problema Notwen

Propusă de: stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București

Notwen a auzit de descoperirile prietenului său de pe Pământ și a decis să studieze și el legile gravitației pe planeta sa. Pentru aceasta a conceput un experiment, care utilizează două drepte (o dreaptă verticală și o dreaptă oblică, înclinată la un unghi oarecare față de orizontală) și un super-măr (care, pentru a simplifica analiza, este considerat punctiform), ca în figură.

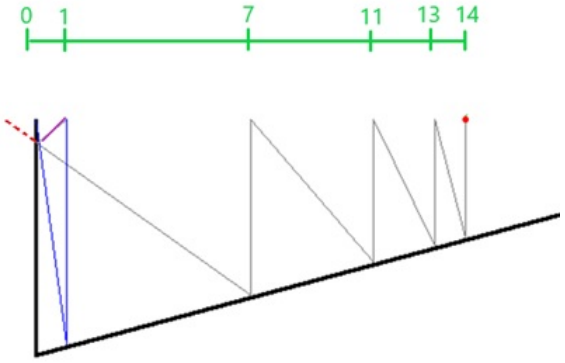


Figura 1. Traiectoria unui super-măr situat inițial la o distanță de 14 cm de dreapta verticală

Super-mărul este lăsat să cadă de la o înălțime mare, de la o distanță de x cm față de dreapta verticală. Super-mărul cade vertical până când întâlnește dreapta înclinată. Când super-mărul se ciocnește de dreapta înclinată acesta sare mult în sus și spre dreapta verticală, deplasându-se astfel cu 1 cm spre dreapta verticală. Apoi, din cauza gravitației, el cade din nou vertical până întâlnește dreapta înclinată. La a doua ciocnire, super-mărul sare mult în sus și spre dreapta verticală, deplasându-se astfel cu 2 cm spre dreapta verticală. A treia oară când se ciocnește cu dreapta înclinată sare mult în sus și spre dreapta verticală, deplasându-se astfel cu 4 cm spre dreapta verticală ș.a.m.d. Notwen a observat că la fiecare ciocnire, exceptând prima, super-mărul se deplasează spre dreapta verticală cu o distanță dublă față de cea de la deplasarea precedentă. Vom numi acest proces oscilație.

La un moment dat super-mărul se ciocnește de dreapta verticală și are loc un recul. Dacă la ultima ciocnire cu dreapta înclinată super-mărul s-a aflat la o distanță de y cm de verticală, urmând să se deplaseze cu z cm, dar y < z, acesta se va ciocni de dreapta verticală și va avea un recul de $z - y$ cm, adică va fi „aruncat" înapoi la distanța $z - y$ cm de dreapta verticală.

Apoi super-mărul își reia mișcarea în același mod, apropiindu-se de verticală la fiecare ciocnire cu dreapta înclinată mai întâi cu 1 cm, apoi cu 2 cm, 4 cm, ș.a.m.d.

Studiind mișcarea super-mărului Notwen a observat că cele două procese (oscilație, recul) alternează până când super-mărul ajunge la distanța de 0 cm de dreapta verticală și se oprește.

În **Figura 1.** este ilustrată mișcarea super-mărului pentru cazul în care experimentul începe de la o distanță față de 14 cm de dreapta verticală. Prima oscilație este ilustrată cu o linie gri: super-mărul se ciocnește succesiv de dreapta înclinată la 14, 13, 11, respectiv 7 cm față de verticală, după care se ciocnește de dreapta verticală și are un recul (ilustrat cu linie roșie) și ajunge la 8 – 7 = 1 cm de dreapta aceasta. Începe al doilea proces de oscilație (ilustrat cu linie albastră), dar după prima ciocnire cu dreapta înclinată se deplasează cu 1 cm spre dreapta verticală, deci ajunge chiar pe dreaptă (la 0 cm de aceasta) și atunci se oprește.

Cerințe

Cunoscând distanța x la care se află super-mărul față de dreapta verticală la începutul experimentului:

1. Determinați numărul de ciocniri ale super-mărului cu dreapta verticală.
2. Determinați numărul de ciocniri ale super-mărului cu dreapta înclinată.

Date de intrare

Fișierul de intrare `notwen.in` conține pe prima linie numărul C reprezentând cerința care trebuie rezolvată (1 sau 2). Pe a doua linie se află numărul natural x cu semnificația din enunț.

Date de ieșire

Fișierul de ieșire `notwen.out` conține o singură linie, pe care este scris numărul determinat pentru cerința C din fișierul de intrare.

Restricții

- $1 \leq x \leq 10^{10000}$

#	Puncte	Restricții
1	12	$C = 1, 1 \leq x \leq 10^{18}$
2	12	$C = 2, 1 \leq x \leq 10^{18}$
3	12	$C = 1, 10^{18} < x \leq 10^{100}$
4	12	$C = 2, 10^{18} < x \leq 10^{100}$
5	17	$C = 1, 10^{100} < x \leq 10^{1000}$
6	17	$C = 2, 10^{100} < x \leq 10^{1000}$
7	9	$C = 1, 10^{1000} < x \leq 10^{10000}$
8	9	$C = 2, 10^{1000} < x \leq 10^{10000}$

Exemple

notwen.in	notwen.out
1 14	2
2 14	5
1 2025	5
2 2025	24
1 12345678901234567890	42
2 12345678901234567890	1492

8.5 Rezolvarea problemei Notwen

Soluții parțiale

Se poate simula operația descrisă în enunț utilizând operații cu numere mari.

Soluția oficială

Observație: Reprezentăm numerele în baza 2 și observăm cum se modifică distanța.

Câteva exemple:

$$\begin{aligned} 14 &= 1110_2 \to 1101_2 \to 1011_2 \to 0111_2 \to -1 \to 1 \\ &= 0001_2 \to 0000_2 \\ &= 0 \end{aligned}$$

$$\begin{aligned} 8 &= 1000_2 \to 0111_2 \to 0101_2 \to 0001_2 \to -7 \to 7 \\ &= 0111_2 \to 0110_2 \to 0100_2 \to 0000_2 \\ &= 0 \end{aligned}$$

$$\begin{aligned} 21 &= 10101_2 \to 10100_2 \to 10010_2 \to 01110_2 \to 00110_2 \to -10 \to 10 \\ &= 01010_2 \to 01001_2 \to 00111_2 \to 00011_2 \to -5 \to 5 \\ &= 00101_2 \to 00100_2 \to 00010_2 \to -2 \to 2 \\ &= 00010_2 \to 00001_2 \to -1 \to 1 \\ &= 00001_2 \to 00000_2 \\ &= 0 \end{aligned}$$

Dacă analizăm câteva numere putem observa o regulă generală. Pornind de la un număr în binar, prima coordonată după ciocnirea cu bara verticală se obține inversând biții numărului (un bit 0 se transformă în 1 și un bit 1 în 0).

Demonstrația acestui fapt este următoarea:

Numerele scăzute din distanță sunt $1, 2, 4, 8, 16, \ldots$. Dar de fapt se scade un prefix din acest șir, până când numărul devine negativ. Șirul de prefixe este $1, 3, 7, 15, 31, \ldots$. Acestea sunt numere de forma $2^k - 1$ unde k este un număr natural nenul. În binar acestea au forma $1_2, 11_2, 111_2, 1111_2, 11111_2, \ldots$.

Primul moment în care super-mărul lăsat să cadă de la distanța x se lovește de bara verticală este dat de k-ul minim pentru care $2^k - 1 \geq x$. Acesta corespunde exact celui mai semnificativ bit din x. Poziția super-mărului după ciocnirea de bara verticală este $-\left(x - \left(2^k - 1\right)\right) = 2^k - 1 - x$.

Deoarece $2^k - 1 = 1111\ldots1_2$ iar $x \leq 2^k - 1$ formula $2^k - 1 - x$ se reduce doar la inversarea biților lui x.

Obținem următoarea soluție:

Convertim numărul în baza 2. Pentru fiecare bit i care diferă de bitul $i + 1$ vom adăuga ceva la răspuns, fie 1 pentru cerința 1, fie $i + 1$ (sau i dacă se indexează de la 1) pentru cerința 2.

Conversia în baza 2

Algoritmul clasic de conversie al unui număr N din baza 10 în baza 2 folosește împărțiri repetate la 2. Resturile obținute sunt scrise în ordine inversă pentru a obține reprezentarea binară a lui N. Acest algoritm, cel puțin în forma aceasta este prea lent.

Pentru a optimiza algoritmul avem 2 posibilități.

1. Stocăm numărul într-o bază mai mare, ușor de calculat din baza 10, mai exact $10^{B_{10}}$. Pentru aceasta vom stoca B_{10} cifre într-un singur număr.
2. Convertim numărul într-o bază mai mare, din care putem ajunge ușor la baza 2, mai exact 2^{B_2}. După ce am făcut această conversie, fiecare „cifră” din această bază va reprezenta B_2 biți.

Soluția oficială folosește ambele optimizări, cu $B_{10} = 9$ și $B_2 = 30$. Astfel convertim un număr din baza 10^9 în baza 2^{30}. Aceste numere sunt apropiate și se poate observa că algoritmul va avea complexitatea $O(C^2)$ unde C este numărul de „cifre” din reprezentarea în baza 10^9 a lui N.

Bonus

Aceste două optimizări îmbunătățesc algoritmul de conversie. Există totuși o limită impusă de reprezentarea numerelor pe calculator, depinzând de dimensiunea maximă a unui număr. Mai exact, conversia poate genera o eroare dacă $2^{B_2} \cdot 10^{B_{10}} - 1$ nu poate fi reprezentat fără probleme pe calculator.

Astfel, dacă ne dorim să creștem B_{10}, s-ar putea ca B_2 să trebuiască să scadă pentru a nu avea probleme.

8.6 Cod-sursă pentru problema Notwen

```
#include<cstdio>
const int NMAX=2000, B2=30;
const int BASE2=1073741824; /*2 la puterea B2*/
const int BASE10=1000000000, CIFMAX=10005, BITSMAX=40000;

int compressed[NMAX], N, B;
char cifre[CIFMAX];
bool bits[BITSMAX];

void conversie_1()
{
    int i, p10=1;
    for(i=0; cifre[i]!='\n' && cifre[i]; ++i);
    for(--i; i>-1; --i)
    {
        compressed[N]+=p10*(cifre[i]-'0');
        p10*=10;
        if(p10==BASE10)
        {
            p10=1;
            ++N;
        }
    }

    if(compressed[N])
        ++N;
}
```

```
int divBaseP2()
{
    long long rest=0;
    int i;

    for(i=N-1; i>-1; --i)
    {
        rest=rest*BASE10+compressed[i];
        compressed[i]=rest/BASE2;
        rest%=BASE2;
    }

    while(N && compressed[N-1]==0)
        --N;

    return rest;
}

void conversie_2()
{
    int i, rest;

    while(N)
    {
        rest=divBaseP2();
        for(i=0; i<B2; ++i)
        {
            bits[B++]=rest%2;
            rest/=2;
        }
    }

    while(B && !bits[B-1])
        --B;
}

void debug_print()
{
    int i;
    for(i=B-1; i>-1; --i)
        printf("%c", bits[i] ? '1' : '0');
}

long long cerinta_1()
{
    bool prev=false;
    int i;
    long long ans=0;

    for(i=B-1; i>-1; --i)
    {
        if(prev!=bits[i])
        {
            prev=bits[i];
            ++ans;
        }
    }
    return ans;
}
```

```
long long cerinta_2()
{
    bool prev=false;
    int i;
    long long ans=0;

    for(i=B-1; i>-1; --i)
    {
        if(prev!=bits[i])
        {
            prev=bits[i];
            ans+=i+1;
        }
    }
    return ans;
}

int main()
{
    FILE* f=fopen("notwen.in", "r"), *g=fopen("notwen.out", "w");
    int C;

    fscanf(f, "%d", &C);
    fgets(cifre, CIFMAX, f);
    fgets(cifre, CIFMAX, f);
    conversie_1();
    conversie_2();
    // debug_print();
    fprintf(g, "%lld\n", C==1 ? cerinta_1() : cerinta_2());
    fclose(f); fclose(g);
    return 0;
}
```

8.7 Problema Program

Propusă de: stud. Răzvan Rotaru, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iași

Mihăiță, elevul talentat al exigentului profesor de muzică Jean Carapace, primește un program de studiu special, care constă în studierea în ordine a N capitole, numerotate de la 1 la N. Capitolul i ($1 \leq i \leq N$) trebuie să fie studiat exact z_i zile consecutive. Studiul capitolului i ($1 \leq i \leq N$) trebuie să se termine cel târziu în ziua t_i. Pentru a finaliza cu succes programul de studiu, Mihăiță trebuie să studieze toate capitolele (se garantează că acest lucru este posibil).

Pe lângă muzică, Mihăiță iubește expedițiile montane. Prietenii îi fac P propuneri, fiecare propunere conținând una sau mai multe expediții. Pentru fiecare expediție se cunoaște intervalul de timp $[a, b]$ în care se desfășoară (începe în ziua a și se termină în ziua b, inclusiv). O propunere se numește *acceptabilă* dacă Mihăiță poate să meargă în toate expedițiile din propunerea respectivă și să finalizeze cu succes programul de studiu.

Cerințe

Se cunosc programul de studiu, precum și propunerile primite:

1. Determinați ziua numerotată cu valoarea maximă în care Mihăiță poate începe programul de studiu astfel încât să-l finalizeze cu succes, în cazul în care nu merge în nicio expediție.
2. Pentru fiecare propunere, determinați numărul maxim de expediții care se suprapun în aceeași zi.
3. Pentru fiecare propunere, verificați dacă este acceptabilă, știind că nicio propunere nu conține expediții care se suprapun.

Date de intrare

Fișierul de intrare `program.in` conține pe prima linie numărul natural C, reprezentând cerința care trebuie rezolvată (1, 2 sau 3). Pe a doua linie se află numărul natural N. Pe a treia linie se află N numere naturale $z_1\ z_2 \ldots z_N$. Pe a patra linie se află N numere naturale $t_1\ t_2 \ldots t_N$. Pe a cincea linie se află numărul natural P. Toate aceste valori au semnificația din enunț. Urmează descrierea celor P propuneri, fiecare fiind descrisă pe câte 3 linii:

- pe prima linie numărul natural M, reprezentând numărul de expediții din propunere;
- pe a doua linie M numere naturale $a_1\ a_2 \ldots a_M$, unde a_i reprezintă ziua de începere a expediției i ($1 \leq i \leq M$);
- pe a treia linie M numere naturale $b_1\ b_2 \ldots b_M$, unde b_i reprezintă ultima zi a expediției i ($1 \leq i \leq M$).

Valorile scrise pe aceeași linie în fișierul de intrare, sunt separate prin câte un singur spațiu.

Date de ieșire

Fișierul de ieșire `program.out` conține o singură linie, pe care se afișează:

- dacă $C = 1$: un număr natural reprezentând răspunsul la cerința 1;
- dacă $C = 2$: P numere naturale separate prin câte un spațiu, unde al i-lea număr reprezintă numărul maxim de expediții din propunerea i ($1 \leq i \leq P$) care se suprapun în aceeași zi;
- dacă $C = 3$: P numere naturale separate prin câte un spațiu, unde al i-lea număr este 1 dacă propunerea i este acceptabilă sau 0 în caz contrar ($1 \leq i \leq P$).

Restricții

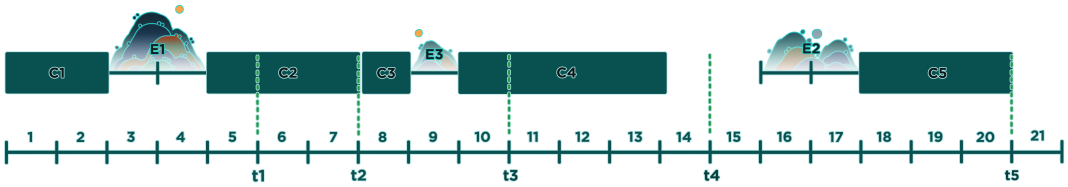
- $1 \leq N, P, M \leq 200\,000$
- $1 \leq z_i, t_i \leq 10^9$, pentru $1 \leq i \leq N$
- $1 \leq a \leq b \leq 10^9$ pentru fiecare expediție din oricare propunere
- $1 \leq Vmax \leq 10^9$, unde $Vmax$ este maximul dintre z_i, t_i, a, b pentru $1 \leq i \leq N$ și a, b din toate expedițiile.
- $1 \leq S \leq 200\,000$, unde S este numărul total de expediții din toate cele P propuneri
- Zilele sunt numerotate începând cu 1.

#	Puncte	Restricții
1	12	$C = 1$, $P = 1$, $1 \leq N \leq 5\,000$
2	9	$C = 1$, $P = 1$, fără alte restricții
3	15	$C = 2$, $N = 1$, $1 \leq Vmax, P \leq 5\,000$
4	17	$C = 2$, $N = 1$, fără alte restricții
5	29	$C = 3$, $1 \leq N, P \leq 5\,000$
6	18	$C = 3$, fără alte restricții

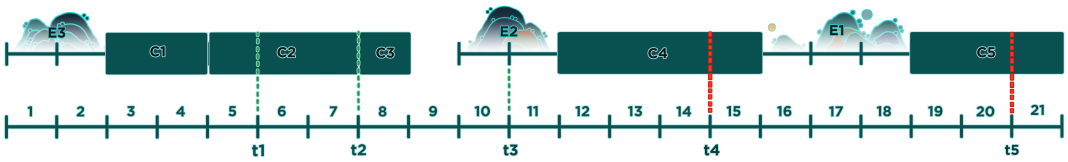
Exemple

program.in	program.out	Explicații
1 5 2 3 1 4 3 5 7 10 14 20 1 1 1 1	3	Cerința este 1. Ziua numerotată cu valoarea maximă în care poate fi început programul de studiu astfel încât să se finalizeze cu succes este 3. Capitolul 1 este studiat în zilele 3, 4. Capitolul 2 este studiat în zilele 5, 6, 7.
2 1 1 1 2 3 3 16 9 4 17 9 6 3 2 10 7 1 6 8 10 16 12 3 14	1 4	Cerința este 2. Pentru prima propunere, numărul maxim de expediții care au loc în aceeași zi este 1, deoarece perioadele nu se suprapun. Pentru a doua propunere răspunsul este 4, deoarece expedițiile 2, 3, 4 și 6 se desfășoară simultan în ziua 10.
3 5 2 3 1 4 3 5 7 10 14 20 2 3 3 16 9 4 17 9 3 16 10 1 18 11 2	1 0	Cerința este 3. Explicația este ilustrată în figurile următoare.

Prima propunere îi permite lui Mihăiță să parcurgă toate capitolele la timp:



A doua propunere nu îi permite lui Mihăiță să parcurgă toate capitolele la timp:



8.8 Rezolvarea problemei Program

Subtaskul 1: $1 \leq N \leq 5000$, $P = 1$, $C = 1$

Se determină ziua maximă de începere a studiului astfel încât toate cursurile să se finalizeze la timp, fără a include excursiile. Deoarece cursurile se desfășoară secvențial, timpul minim de finalizare al fiecărui curs se calculează prin însumarea duratelor cursurilor anterioare și a celui curent. Folosind tehnica sumelor parțiale, se obține pentru fiecare curs timpul minim necesar finalizării acestuia. Acest timp se compară cu termenul limită al cursului, iar diferența (termenul limită minus timpul minim de finalizare) indică cu cât poate fi amânată începerea studiului fără a încălca restricțiile de timp. Pentru determinarea minimului, se compară toate perechile de diferențe. Ziua maximă de start se stabilește prin identificarea valorii minime a acestor diferențe, aceasta fiind numărul maxim de zile în care Mihăiță poate să nu învețe, la care adunăm 1.

Complexitate: O(N^2).

Observație: o soluție care încearcă să fixeze inițial, în mod consecutiv, ziua în care Mihăiță începe parcurgerea cursurilor și verifică pentru fiecare curs dacă poate fi finalizat la timp, căutând valoarea maximă a acestei zile de start pentru care condiția este satisfăcută, obține jumătate din punctajul pentru acest subtask. Complexitate: O($Vmax * N$).

Subtaskul 2: Fără restricții suplimentare, $C = 1$, $P = 1$

Pentru a găsi minimul dintre diferențe este suficientă o singură parcurgere, în care reținem minimul de zile necesare, într-o variabilă auxiliară. O altă abordare posibilă pentru $c = 1$ este să încercăm să poziționăm cursurile cât mai aproape de termenul limită, aflând astfel ziua maximă în care Mihăiță se poate apuca de studierea cursurilor. Pentru această abordare trebuie să parcurgem cursurile de la ultimul la primul și să ne asigurăm că perioadele în care fixăm parcurgerea cursurilor nu se intersectează. Complexitate: O(N).

Subtaskul 3: $1 \leq P$, $Vmax \leq 5000$, $N = 1$, $C = 2$

O excursie va fi asimilată cu un interval închis. Pentru a determina numărul maxim de excursii desfășurate simultan într-o anumită zi, se aplică algoritmul „Șmenul lui Mars”. Metoda utilizează un vector de frecvență în care, pentru fiecare excursie, se marchează extremitatea de început cu +1 și cu -1 poziția imediat următoare extremității de final. După parcurgerea vectorului de frecvență în ordine și calcularea sumei parțiale, se obține numărul de excursii active pentru fiecare

zi. Numărul maxim obținut la un moment dat indică numărul maxim de excursii care se suprapun în aceeași zi. Complexitate: O($P * Vmax$).

Subtaskul 4: Fără restricții suplimentare, $C = 2$, $N = 1$

În acest subtask, extremitățile intervalelor excursiilor pot avea valori foarte mari, ceea ce face imposibilă utilizarea unui vector de frecvență. Pentru a rezolva această problemă se utilizează o tehnică de „liniarizare" a extremităților. Se extrag toate extremitățile intervalelor și se etichetează conform tipului lor: extremitățile de început sunt marcate cu +1, iar cele de final cu -1. Aceste extremități se sortează în ordine cronologică, iar, în caz de egalitate, se acordă prioritate extremităților de început (pentru a reflecta corect începerea unui interval înainte ca altul care are un timp de final identic să se încheie). Se parcurge lista sortată:

- dacă extremitatea curentă este una început al unui interval, creștem numărul de excursii care se desfășoară la momentul curent;
- dacă extremitatea curentă este una de final al unui interval, comparăm numărul de excursii care se desfășoară simultan la momentul curent (numărul de intervale deschise care nu au fost închise) cu maximul, apoi scădem cu 1 numărul de excursii care se desfășoară la momentul curent.

Valoarea maximă obținută indică numărul maxim de excursii care se suprapun într-o anumită zi. Complexitate: O($S * logS$).

Subtaskul 5: $1 \leq N, P \leq 5000$, $C = 3$

Soluția se bazează pe un algoritm de tip interclasare: se parcurg simultan lista capitolelor și lista excursiilor, ambele sortate crescător. Folosim o variabilă care reține prima zi disponibilă pentru studiu. Pentru fiecare excursie se analizează perioada liberă de studiu dinaintea datei de start a excursiei, pentru a verifica dacă aceasta permite finalizarea completă a capitolului curent. Dacă intervalul liber este suficient pentru a termina cursul, se programează acel capitol și se trece la capitolul următor, continuând evaluarea în cadrul aceluiași interval liber. Dacă intervalul nu este suficient de mare, se actualizează data disponibilă pentru studiu la ziua imediat următoare zilei în care se termină excursia, iar încercarea de a finaliza același capitol se reia cu noua perioadă liberă. Propunerea este acceptată doar dacă, pentru fiecare curs, ziua de finaliza nu depășește termenul limită. Complexitate: O($P * N$), deoarece pentru fiecare propunere iterăm prin fiecare capitol.

Subtaskul 6 – Fără restricții suplimentare, $C = 3$

Se extinde metoda de la Subtaskul 5 și se optimizează determinarea numărului maxim de cursuri ce pot fi parcurse într-un interval liber prin utilizarea căutării binare pe vectorul de sume parțiale al duratelor cursurilor (calculat și la Subtaskul 1). Se precalculează, de asemenea, un vector auxiliar care, pentru fiecare curs, reține minimul diferenței dintre termenul limită și numărul de zile necesare studiului cursului curent și al celor ce urmează, facilitând astfel verificarea rapidă a constrângerilor. Pentru fiecare propunere de excursii, intervalele se sortează crescător după extremitatea inițială. Parcurgem intervalele libere dinaintea fiecărei excursii și, pentru fiecare interval liber, căutăm binar pe vectorul de sume parțiale pentru a identifica rapid numărul maxim de cursuri finalizabile înainte de începerea excursiei curente. Dacă nu se pot programa toate cursurile disponibile în intervalul respectiv, se trece la următoarea excursie, se actualizează perioada de studiu disponibilă și se repetă verificarea pentru cursurile rămase, asigurându-se totodată că fiecare termen limită este respectat. Complexitate: O($SlogN$).

8.9 Cod-sursă pentru problema Program

```
#include <fstream>
#include <algorithm>
using namespace std;
ifstream cin ("program.in");
ofstream cout ("program.out");
int s[200008],spatiu[200008],min_spatiu[200008];
int i,m,n,p,j,c,suma,minn;
// retinem pentru fiecare capitol numarul de zile necesare si termenul limita
struct capitol
{
    int nr_zile,term_lim;
} v[200008];

// retinem pentru fiecare excursie capetele intervalului acesteia
struct excursie
{
    int a,b;
} grup[200008];

// retinem pentru fiecare capat pozitia si valoarea sa
struct punct
{
    int poz,val;
} pct[400008];

// functie de comparare care defineste modul de sortare a capetelor fiecarei excursii
bool compar_pct(punct p1,punct p2)
{
    // sortam dupa pozitie, iar in caz de egalitate, prioritizam punctele de inceput
    return p1.poz<p2.poz || p1.poz==p2.poz && p1.val<p2.val;
}
// functie de comparare care defineste modul de sortare a excursiilor
bool compar_gr(excursie e1,excursie e2)
{
    // sortam excursiile crescator dupa capatul din stanga
    return e1.a<e2.a;
}

// functie care citeste un grup de excursii
void citire_grup()
{
    cin>>m;
    for(int j=1; j<=m; j++)
        cin>>grup[j].a;
    for(int j=1; j<=m; j++)
        cin>>grup[j].b;
}
// functie care returneaza numarul maxim de excursii care au loc in acelasi timp
int int_seg()
{
    int j,cnt=0,maxx=0;
    for(j=1; j<=m; j++)
    {
        pct[j*2-1]= {grup[j].a,1};
        pct[j*2]= {grup[j].b+1,-1};
    }
    sort(pct+1,pct+m*2+1,compar_pct);
    for(j=1; j<=m*2; j++)
    {
```

```
        cnt+=pct[j].val;
        maxx=max(maxx,cnt);
    }
    return maxx;
}

// functie care returneaza numarul maxim de excursii care pot fi puse
// inainte de parcurgerea cursului aferent valorii val
int caut_bin(int val)
{
    int st=1,dr=n,mij,ras=0;
    while(dr>=st)
    {
        mij=(dr+st)/2;
        if(s[mij]<val)
        {
            ras=mij;
            st=mij+1;
        }
        else
        {
            dr=mij-1;
        }
    }
    return ras;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin>>c;
    cin>>n;
    for(i=1; i<=n; i++)
        cin>>v[i].nr_zile;
    for(i=1; i<=n; i++)
        cin>>v[i].term_lim;
    if(c==1)
    {
        cin>>p;
        minn=1e9;
        for (i=1; i<=p; i++) citire_grup();
        for (i=1; i<=n; i++)
        {
          suma+=v[i].nr_zile;
          // numarul total de zile necesare pentru a parcurge primele i cursuri
          minn=min(minn,v[i].term_lim-suma);
          // diferenta dintre termenul limita al cursului i
          // si numarul total de zile necesare pentru a parcurge primele i cursuri
        }
        if (minn<0)
            cout<<-1;
        else
            cout<<minn+1;
    }
    else if (c==2)
    {
        cin>>p;
        for (i=1; i<=p; i++)
        {
          citire_grup();
```

```
          cout<<int_seg()<<' ';
        }
    }
    else
    {
      for (i=1; i<=n; i++)
      {
          s[i]=s[i-1]+v[i].nr_zile;
          spatiu[i]=v[i].term_lim-s[i];
      }
      min_spatiu[n]=spatiu[n];
      for (i=n-1; i>=1; i--)
          min_spatiu[i]=min(min_spatiu[i+1],spatiu[i]);
      // min_spatiu[i] reprezinta numarul maxim de zile in care Mihaita
      // poate sa nu studieze, inainte de a parcurge cursul i
      cin>>p;
      for (i=1; i<=p; i++)
      {
        citire_grup();
        int ok=1;
        sort(grup+1,grup+m+1,compar_gr);
        int loc_liber=0;
        for (j=1; j<=m; j++)
        {
           int poz=caut_bin(grup[j].a-loc_liber);
           if (poz==n)
           // daca am reusit sa parcurgem toate cursurile,
           // inseamna ca propunerea de excursii este acceptabila
               break;
           loc_liber=grup[j].b-s[poz];
           // variabila "loc_liber" reprezinta cate zile a stat Mihaita
           // fara sa studieze pana in momentul actual
           if (min_spatiu[poz+1]-loc_liber<0)
           // verificam daca vom putea duce toate cursurile viitoare la bun sfarsit
           // (sa respecte termenul limita)
           {
              ok=0;
              break;
           }
        }
        cout<<ok<<" ";
    }
  }
  return 0;
}
```

Capitolul 9

Baraj selecție lot juniori ONI 2025

9.1 Problema Joc

Propusă de: prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași

Pentru a îmbunătăți aptitudinile logico-matematice ale elevilor săi, profesorul Vasile a implementat un joc. Pe ecranul principal al jocului se afișează un șir de N scaune, numerotate de la stânga spre dreapta începând cu 1, pe fiecare scaun fiind așezat câte un copil. Fiecare copil poartă un tricou pe care este scris, de asemenea, câte un număr de la 1 la N. Numerele de pe tricouri sunt distincte și sunt scrise pe spate, deci nu sunt vizibile.

T

	1	2	3
1	1	2	2
2		2	2
3			3

Scopul jocului este de a descoperi numărul scris pe tricoul fiecărui copil. Pentru aceasta, pe ecran mai este afișat un triunghi de numere T, care ne dă informații ajutătoare. Triunghiul arată ca o matrice în care liniile sunt numerotate de sus în jos de la 1 la N, iar coloanele de la stânga la dreapta de la 1 la N. Numărul scris în triunghi pe linia i și coloana j ($1 \leq i \leq j \leq n$) reprezintă numărul scaunului pe care stă copilul având cel mai mic număr pe tricou dintre toți copiii situați pe scaune cu numere cuprinse între i și j (inclusiv i și j). Observați că pozițiile din triunghi de pe linia i și coloana j cu $1 \leq j < i \leq N$ nu sunt completate.

Numim *soluție* o succesiune de N numere naturale distincte cuprinse între 1 și N care ar putea fi scrise, în ordine, de la stânga la dreapta, pe tricourile celor N copii, astfel încât informațiile din triunghiul de numere să fie corecte. Două soluții sunt considerate distincte dacă există cel puțin un copil pentru care numărul scris pe tricoul său în cele două soluții diferă.

Cerințe

Cunoscând numărul de copii și triunghiul de numere:

1. determinați o soluție posibilă; dacă există mai multe soluții posibile se va afișa cea mai mică din punctul de vedere lexicografic;

2. determinați numărul de soluții posibile.

Date de intrare

Fișierul de intrare `joc.in` conține pe prima linie numărul natural C reprezentând cerința care trebuie să fie rezolvată (1 sau 2). Pe linia a doua se află numărul natural N cu semnificația din enunț. Pe următoarele N linii se află numerele din triunghi. Pe linia i dintre cele N sunt scrise $N - i + 1$ numere separate prin câte un spațiu, reprezentând numerele de pe linia i din triunghi, situate pe coloanele i, $i + 1, \ldots N$.

Date de ieșire

Fișierul de ieșire `joc.out` conține o singură linie.
Dacă $C = 1$ linia conține N numere naturale distincte cuprinse între 1 și N, separate prin câte un spațiu, reprezentând soluția cea mai mică din punct de vedere lexicografic determinată pentru cerința 1.
Dacă $C = 2$ linia conține numărul de soluții posibile determinat pentru cerința 2.

Restricții

- $1 \leq N \leq 1\,000$
- Spunem că șirul a_1, a_2, $\ldots a_N$ este mai mic din punctul de vedere lexicografic decât șirul b_1, b_2, $\ldots b_N$ dacă există k $(1 \leq k \leq N)$, astfel încât $a_i = b_i$, pentru orice $1 \leq i < k$ și $a_k < b_k$.
- Se garantează că, pentru datele de test, există cel puțin o soluție.

#	Puncte	Restricții	
1	9	$C = 1$,	$1 \leq N < 10$
2	9	$C = 2$,	$1 \leq N < 10$
3	22	$C = 1$,	$10 \leq N \leq 1000$
4	24	$C = 2$,	$10 \leq N < 28$
5	36	$C = 2$,	fără restricții suplimentare

Exemple

joc.in	joc.out
1 3 1 2 2 2 2 3	2 1 3
2 3 1 2 2 2 2 3	2

Explicație

O soluție posibilă este 2 1 3.
Pe secvența de scaune 1..1 copilul cu număr minim pe tricou este pe scaunul 1.

Pe secvența de scaune 1..2 copilul cu număr minim pe tricou este pe scaunul 2
Pe secvența de scaune 1..3 copilul cu număr minim pe tricou este pe scaunul 2
Pe secvența de scaune 2..2 copilul cu număr minim pe tricou este pe scaunul 2
Pe secvența de scaune 2..3 copilul cu număr minim pe tricou este pe scaunul 2
Pe secvența de scaune 3..3 copilul cu număr minim pe tricou este pe scaunul 3
O altă soluție posibilă este 3 1 2, dar 2 1 3 este mai mică din punct de vedere lexicografic.

9.2 Rezolvarea problemei Joc

Soluția 1. Cerința 1

Vom reține elementele triunghiului într-o matrice T cu N linii și N coloane, deasupra diagonalei principale.

Așezarea elevilor pe scaune reprezintă o permutare de ordin N, pe care o vom reconstitui într-un vector *sol* cu N elemente, numerotate de la 1 la N. Pentru a reconstitui permutarea minimă din punct de vedere lexicografic:

- $T[1][N]$ reprezintă poziția elementului minim (adică 1); deci, vom plasa în permutare elementul 1 pe poziția $T[1][N]$.
- În stânga elementului 1 se vor afla $T[1][N] - 1$ elemente, iar în dreapta sa celelalte $N - T[1][N]$; pentru a obține permutarea minimă din punct de vedere lexicografic, vom alege să plasăm în stânga cele mai mici $T[1][N] - 1$ elemente, adică valorile $2, 3, \ldots, T[1][N]$, iar în dreapta celelalte.
- Am redus astfel problema la rezolvarea a două subprobleme de același tip, pe care le vom rezolva recursiv, ele fiind independente.

Putem formula o subproblemă la modul general astfel: „să se reconstituie secvența din permutarea *sol* de la poziția *st*, până la poziția *dr* (inclusiv), știind că în această secvență se vor plasa valori distincte cuprinse între valoarea *minim* și valoarea *maxim*".

Funcția *reconstituire* implementează recursiv acest procedeu

Soluția 1. Cerința 2

Pentru a număra câte modalități de reconstituire a permutării există procedăm într-un mod similar:

- plasăm elementul minim pe poziția $pozmin = T[st][dr]$;
- au rămas $dr - st$ elemente din care trebuie să alegem $pozmin - st$ elemente pentru a fi plasate pe pozițiile $st \ldots pozmin - 1$; aceasta se poate face în combinări de $dr - st$ luate câte $pozmin - st$ moduri (în dreapta fiind automat plasate elementele rămase)

Functia *numarare* implementează acest procedeu.

Se obține un produs de combinări care trebuie calculat pe numere mari; pentru aceasta vom reține într-un vector p descompunerea în factori primi a acestui produs ($p[x]$=puterea factorului prim x în produsul combinărilor).

Pentru a calcula rezultatul final este suficient să înmulțim la rezultat factorii primi la puterea corespunzătoare, fiind necesară doar o funcție de înmulțire a unui număr mare cu un număr mic.

Soluția 2 - prof. Adrian Panaete

Pentru a evita abordarea recursivă de mai sus se poate simula recursivitatea utilizând o stivă în care se vor memora intervalele de care se ocupă fiecare apel recursiv. Observăm că în ambele cerințe recursivitatea funcționează în modul următor

- un apel recursiv se va face pe un interval de indici $[st, dr]$;
- apelul izolează o poziție mi a intervalului $[st, dr]$ pe care o tratează individual (în moduri diferite in funcție de cerință)
- se apelează funcția recursivă pe intervalul $[st, mi - 1]$ (dacă acesta nu este vid)
- se apelează funcția recursivă pe intervalul $[mi + 1, dr]$ (dacă acesta nu este vid).

Putem observa că fiecare procesare se referă la intervale de indici $[st, dr]$ și recursivitatea va procesa aceste intervale într-o anumită ordine.

În loc să folosim o funcție recursivă putem folosi o stivă în care să adăugăm și/sau să eliminăm intervale astfel încât să procesăm toate intervale din soluția recursivă, iar procesarea să se realizeze în aceeași ordine ca în soluția recursivă.

Pentru a realiza acest lucru se folosește următorul algoritm:

- Adăugăm în stivă intervalul $[1, n]$.
- Cât timp există elemente în stivă:
  - Scoatem intervalul $[st, dr]$ din vârful stivei.
  - Identificăm și procesăm poziția mi.
  - Adăugăm în stivă intervalul $[mi + 1, dr]$ (dacă nu este vid).
  - Adăugăm în stivă intervalul $[st, mi - 1]$ (dacă nu este vid).

În rest, toate detaliile de implementare sunt la fel ca în soluția recursivă.

9.3 Cod-sursă pentru problema Joc

```
#include <fstream>
#define NMAX 1002
#define LGMAX 10000
using namespace std;
ifstream fin("joc.in");
ofstream fout("joc.out");
typedef int NrMare[LGMAX];
int n, cerinta;
NrMare rez;
int lgrez;
int T[NMAX][NMAX];
int sol[NMAX];
int p[NMAX];

void citire();
void reconstituire(int st, int dr, int minim, int maxim);
void combinari(int n, int m);
void numarare(int st, int dr);
void prod(NrMare a, int lga, int x, NrMare p, int& lgp);
void descompunere(int st, int dr, int semn);
int main()
{int i, j;
 citire();
 if (cerinta==1)
    {reconstituire(1,n,1,n);
     for (i=1; i<n; i++) fout<<sol[i]<<' ';
```

```
     fout<<sol[n]<<'\n';
    }
    else
    {
     numarare(1,n);
     rez[0]=1; lgrez=1;
     for (i=2; i<NMAX; i++)
         for (j=0; j<p[i]; j++)
              prod(rez,lgrez,i,rez,lgrez);
     for (i=lgrez-1; i>=0; i--) fout<<rez[i];
     fout<<'\n';
    }
 //fout<<~(0ull);
 return 0;
}

void citire()
{int i, j;
 fin>>cerinta>>n;
 for (i=1; i<=n; i++)
     for (j=i; j<=n; j++)
         fin>>T[i][j];
}

void reconstituire(int st, int dr, int minim, int maxim)
{int pozmin;
 if (st<=dr)
     {pozmin=T[st][dr];
      sol[pozmin]=minim;
      reconstituire(st,pozmin-1,minim+1, minim+pozmin-st);
      reconstituire(pozmin+1,dr, minim+pozmin-st+1,maxim);
     }
}

void numarare(int st, int dr)
{
 if (st<dr)
    {
     int pozmin=T[st][dr];
     combinari(dr-st,pozmin-st);
     numarare(st,pozmin-1);
     numarare(pozmin+1,dr);
    }
}

void combinari(int n, int m)
{
 if (m>0 && m<n)
    {descompunere(n-m+1,n,1);
     descompunere(2,m,-1);
    }
}

void descompunere(int st, int dr, int semn)
{int k, d, x;
 for (k=st; k<=dr; k++)
      {d=2; x=k;
       while (d*d<=x)
             {
               while (x%d==0)
                     {p[d]+=semn;
```

```
                    x/=d;}
               d++;
              }
       if (x>1) p[x]+=semn;
      }
}

void prod(NrMare a, int lga, int x, NrMare p, int& lgp)
{int t = 0, val, i;
 for (i=0; i<lga; i++)
     {
       val = a[i] * x + t;
       p[i] = val %10;
       t = val / 10;
    }
 lgp = lga;
 while (t)
      {
       p[lgp] = t%10;
       lgp++;
       t = t/10;
    }
}
```

9.4 Problema Succes

Propusă de: stud. Alin Răileanu, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iași

Se consideră șirul $S = S_1, S_2, \ldots S_N$ format din N mulțimi de numere naturale cuprinse între 1 și M. De asemenea, se consideră două șiruri de câte M numere întregi $A = A_1, A_2, \ldots A_M$ și $B = B_1, B_2, \ldots B_M$.

Numim secvență de mulțimi (i, j) $(1 \leq i \leq j \leq N)$ succesiunea de mulțimi $S_i, S_{i+1}, \ldots S_j$.

Pentru o secvență de mulțimi (i, j) $(1 \leq i \leq j \leq N)$, se determină **factorul de succes** pe baza șirului A, respectiv **factorul de insucces**, pe baza șirului B în modul următor:

1. se efectuează reuniunea mulțimilor din secvența de mulțimi (i, j);
2. **factorul de succes** al secvenței de mulțimi (i, j) este suma valorilor din șirul A situate pe pozițiile date de elementele reuniunii;
3. **factorul de insucces** al secvenței de mulțimi (i, j) este suma valorilor din șirul B situate pe pozițiile date de elementele reuniunii.

O secvență (i, j) $(1 \leq i \leq j \leq N)$ este **câștigătoare** dacă îndeplinește următoarele condiții:

1. **factorul de insucces** al secvenței este cel mult egal cu un număr natural K dat;
2. **factorul de succes** al secvenței este cel mai mare dintre factorii de succes corespunzători tuturor secvențelor ce respectă condiția 1.

Cerințe

Determinați **factorul de succes** al unei secvențe **câștigătoare**.

Date de intrare

Fișierul de intrare `succes.in` conține pe prima linie numerele naturale N, M, K. Pe a doua linie din fișier se găsesc M numere întregi, reprezentând elementele șirului A. Pe a treia linie din fișier se găsesc M numere întregi, reprezentând elementele șirului B. Pe ultimele N linii sunt descrise cele N mulțimi din șirul S, câte o mulțime pe o linie. O linie care descrie o mulțime conține nr, numărul de elemente din mulțime, urmat de cele nr elemente ale mulțimii. Valorile scrise pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `succes.out` conține o singură linie pe care este scris factorul de succes al unei secvențe câștigătoare.

Restricții

- $1 \leq N \leq 200\,000$
- $1 \leq M \leq 100$
- Numărul total de elemente din cele N mulțimi este $\leq 1\,000\,000$
- $0 \leq K, |A_i|, |B_i| \leq 1\,000\,000\,000$, pentru $1 \leq i \leq M$
- Se garantează că există cel puțin o secvență câștigătoare.

#	Puncte	Restricții
1	15	$1 \leq N \leq 100$
2	20	$100 < N \leq 1\,000$
3	21	Numerele din șirurile A și B sunt pozitive.
4	44	Fără restricții suplimentare

Exemple

succes.in	succes.out	Explicații
4 3 3 3 2 -2 1 2 1 3 1 2 3 2 1 2 1 1 2 3 2	5	$N = 4$, $M = 3$, $K = 3$. O secvență **câștigătoare** este $(2, 3)$. Reuniunea mulțimilor pentru secvența $(2, 3)$ este {1,2}U{1}={1,2}. Factorul de insucces pentru secvența $(2, 3)$ este $B_1 + B_2 = 1 + 2 = 3 \leq K$. Factorul de succes pentru secvența $(2, 3)$ este $A_1 + A_2 = 3 + 2 = 5$, valoare maximă pentru toate secvențele pentru care factorul de insucces este $\leq K$.

9.5 Rezolvarea problemei Succes

Pentru început, vom defini funcțiile:

- sc(i, j) - factorul de succes al secvenței delimitate de indicii i și j
- insc(i, j) - factorul de insucces al secvenței delimitate de indicii i și j
- $U(i, j)$ - mulțimea obținută prin reuniunea mulțimilor din secvența delimitată de indicii i și j.

Subtask 1 - $1 \leq N \leq 100$ - 12 puncte

Se vor testa toate secvențele șirului de mulțimi prin selectarea în manieră brută a celor două capete i și j $(1 \leq i \leq j \leq N)$, iar apoi prin calcularea celor 2 valori $sc(i, j)$ și $insc(i, j)$.

Cei doi factori pot fi calculați în complexitate timp $O(N \times M)$, iar selectarea tuturor secvențelor în $O(N^2)$.

Complexitatea finală va fi $O(N^3 \times M)$.

Subtask 2 - $100 < N \leq 1000$ - 13 puncte

Similar primului subtask, se vor testa toate secvențele șirului de mulțimi prin selectarea în manieră brută a celor două capete i și j $(1 \leq i \leq j \leq N)$, iar cei 2 factori se pot actualiza în $O(M)$ la fiecare schimbare de capăt dreapta.

Complexitatea finală va fi $O(N^2 \times M)$.

Subtask 3 - Numerele din A și B sunt pozitive - 20 puncte

Pentru acest caz, *sc* și *insc* vor fi crescătoare pentru un capăt stânga setat (1).

Totodată, faptul că cele 2 șiruri au doar numere pozitive garantează că $insc(i, j) \leq insc(i+1, j)$.

Deci, dacă $insc(i, j) \leq K$, atunci și $insc(i+1, j) \leq K$ (2).

Din (1) și (2) rezultă că putem aplica tehnica **„Two Pointers”** pentru acest caz, optimizând soluția de la subtask-ul 2.

Complexitatea finală va fi $O(N \times M)$.

Subtask 4 - Restricții inițiale - 55 puncte

Spre deosebire de subtask-ul precedent, funcțiile *sc* și *insc* nu mai sunt monotone.

Totuși, funcția U este monotonă pentru un capăt stânga setat, întrucât operația de reuniune este monotonă (1).

Cum cardinalul maxim al unei mulțimi este M și (1), rezultă că funcția U va avea cel mult M schimbări de rezultat pentru un capăt stânga setat.

Cum funcțiile *sc* și *insc* își schimbă rezultatul doar atunci când și funcția U își schimbă rezultatul, o bună optimizare pentru soluția de la subtask-ul 2 este să parcurgem pentru un capăt stânga setat doar capetele dreapta care aduc o schimbare de rezultat pentru funcția U.

Putem observa că pentru un indice i, funcția $U(i, j)$ își modifică rezultatul doar atunci când mulțimea de la poziția j conține un element care nu se găsește în niciuna dintre mulțimile de la i la $j-1$.

Astfel, putem parcurge mulțimile de la N la 1 și să reținem tabloul *last* cu semnificația: $last[x] = \begin{cases} j_{(i \leq j \leq N)}, & j \text{ este cel mai mic indice al unei mulțimi parcurse care conține elementul } x \\ N+1, & \text{dacă nu există un astfel de indice.} \end{cases}$

În continuare, la fiecare iterație vom actualiza valorile din *last*, apoi vom parcurge numerele de la 1 la M în ordinea crescătoare a valorilor din *last* și vom testa rezultatele *sc* și *insc*, având în vedere doar stările valide.

Pentru parcurgerea în ordinea dorită, valorile pot fi sortate sau se poate utiliza următorul „trick”:

- punem la început în sortare toate valorile care nu se găsesc în mulțimea de la indicele curent, în ordinea în care se găseau în sortarea finală de la indicele precedent
- punem la final în sortare elementele mulțimii de la indicele curent.

Pentru soluția cu sortare, complexitatea timp va fi $O(N \times M \times log(M))$, iar în urma optimizării, complexitatea obținută va fi $O(N \times M)$.

Mai există și soluții $O(N \times M \times log(N))$ bazate pe precalculări și căutare binară pe rezultat.

9.6 Cod-sursă pentru problema Succes

```
#include <fstream>
const int MMAX=105;
#define int long long

using namespace std;
ifstream cin("succes.in");
ofstream cout("succes.out");

int a[MMAX], b[MMAX], poz[MMAX];
int ev[MMAX];
```

```
int n, m, k;

signed main()
{
    int i, j, lg, elem, suma, sumb, ans=-1e18;
    cin>>n>>m>>k;
    for(i=1; i<=m; i++) cin>>a[i];
    for(i=1; i<=m; i++) cin>>b[i];
    for(i=1; i<=m; i++) ev[i]=i;
    for(i=1; i<=n; i++)
    {
        cin>>lg;
        for(j=1; j<=lg; j++)
        {
            cin>>elem;
            poz[elem]=i;
        }
        int st=1, dr=m;
        for(j=1; j<=m; j++) ///punem la inceput toate elementele
        {
            if(poz[ev[j]]!=i) ///care nu apar in multimea i
            {
                ev[st++]=ev[j];
            }
        }
        for(j=1; j<=m; j++) ///punem la final toate elementele
        {
            if(poz[j]==i) ///care apar in multimea i
            {
                ev[dr--]=j;
            }
        }
        sumb=suma=0;
        for(j=m; j>=1; j--) ///formam multimile "partiale"
        {
            if(!poz[ev[j]]) break;
            sumb+=b[ev[j]];
            suma+=a[ev[j]];
            if(sumb<=k && poz[ev[j]]!=poz[ev[j-1]]) ans=max(ans, suma);
        }
    }
    cout<<ans<<'\n';
    return 0;
}
```

9.7 Problema Vnoroc

Propusă de: stud. Victor Botnaru, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie POLITEHNICA București

Pentru avea succes la Olimpiada de Jocuri pe Internet (OJI), Vasilică a cumpărat de la Baba Yaga un talisman norocos. Talismanul norocos este un șir care îndeplinește următoarele două condiții:

- toate elementele șirului sunt cifre mai mici sau egale cu magicul 7;
- oricare două elemente aflate pe poziții consecutive în șir au cel puțin un divizor comun strict mai mare decât 1.

De exemplu, (7, 7, 7), (6, 0, 0, 4, 0) și (2, 6, 3) sunt talismane norocoase, dar (1, 2, 3), (2, 4, 7) și (5, 6, 5) nu sunt.

Cel mai mare rival al lui Vasilică, pe nume Acilisav, a aflat de planul lui de a folosi magie pentru a câștiga competiția de jocuri și s-a furișat în noaptea de dinainte de OJI în casa lui Vasilică și i-a modificat talismanul, adăugând cifre mai mici sau egale cu 7.

Vasilică vrea să elimine cifre din șir, astfel încât șirul să redevină un talisman norocos.

Cerințe

Cunoscând șirul V modificat de Acilisav:

1. determinați numărul minim de cifre care trebuie să fie eliminate din șirul V, astfel încât acesta să devină talisman norocos;
2. determinați talismanul norocos minim lexicografic, care se obține eliminând din șirul V un număr minim de cifre.

Date de intrare

Fișierul de intrare `vnoroc.in` conține pe prima linie numerele naturale C și N, reprezentând cerința care trebuie să fie rezolvată (1 sau 2), respectiv numărul de elemente din șirul V. Pe a doua linie se află N cifre separate prin câte un spațiu, reprezentând elementele șirului V.

Date de ieșire

Fișierul de ieșire `vnoroc.out` conține o singură linie, pe care este scris:

- dacă $C = 1$, un număr natural nr, reprezentând numărul minim determinat pentru cerința 1;
- dacă $C = 2$, $(N - nr)$ cifre separate prin câte un spațiu, reprezentând talismanul norocos minim lexicografic, determinat pentru cerința 2.

Restricții

- $2 \leq N \leq 10^6$
- Cifrele din șirul V sunt mai mici sau egale cu 7.
- Spunem că șirul a_1, a_2, $\ldots a_N$ este mai mic strict din punctul de vedere lexicografic decât șirul b_1, b_2, $\ldots b_N$ dacă există k $(1 \leq k \leq N)$, astfel încât $a_i = b_i$, pentru orice $1 \leq i < k$ și $a_k < b_k$.

#	Puncte	Restricții
1	6	$C = 1$, toate cifrele sunt prime
2	12	$C = 1$, $1 \leq N \leq 100$
3	12	$C = 1$, $100 < N \leq 1000$
4	14	$C = 1$, $1000 < N \leq 10^6$
5	6	$C = 2$, toate cifrele sunt prime
6	16	$C = 2$, $1 \leq N \leq 100$
7	16	$C = 2$, $100 < N \leq 1000$
8	18	$C = 2$, $1000 < N \leq 10^6$

Exemple

vnoroc.in	vnoroc.out	Explicații
1 5 4 4 3 6 2	1	Pentru ca șirul să devină norocos, este suficient să eliminăm cifra 3.
2 5 4 4 3 6 2	4 4 6 2	Singura soluție care se poate obține eliminând un număr minim de cifre (1) este 4 4 6 2.
2 6 5 7 5 7 5 7	5 5 5	Numărul minim de cifre care trebuie să fie eliminate este 3. Se pot obține două talismane norocoase prin eliminarea a câte 3 cifre: (5, 5, 5) și (7, 7, 7). Se afișează talismanul norocos minim lexicografic 5 5 5.
2 9 2 3 5 3 7 0 0 5 1	3 3 0 0 5	Numărul minim de cifre care trebuie să fie eliminate este 4. Se poate obține un singur talisman norocos (3, 3, 0, 0, 5) prin eliminarea a 4 cifre.

9.8 Rezolvarea problemei Vnoroc

Soluția 1

Observații inițiale:

- Numărul 1 este singurul număr natural care nu are divizori mai mari decât 1, prin urmare toate valorile egale cu 1 vor fi eliminate din șir de la început (și contorizate ca elemente eliminate).
- Numărul 0 este singurul număr natural care este divizibil cu orice alt număr natural nenul. Vom partiționa șirul dat în secvențe de valori care nu conțin zerouri și vom rezolva problema pentru aceste secvențe, zerourile fiind „punți de legătură" între soluțiile obținute pentru aceste secvențe (utilizând zerourile vom putea concatena aceste soluții, inserând zerourile pe pozițiile corespunzătoare).

Vom analiza în continuare secvențe de valori > 1.

- Dacă secvența conține doar cifrele 2, 3, 5, 7 (cifre prime), oricare două elemente diferite nu au divizori comuni diferiți de 1. Numărăm câte cifre de 2, 3, 5 respectiv 7 avem în șirul

inițial. Soluția va fi alcătuită din cifra care apare de cele mai multe ori. În caz de egalitate, luăm cifra cea mai mică, pentru a face șirul minim lexicografic.
- Dacă secvența conține doar cifrele 2, 3, 4, 5, 7, soluția problemei nu se schimbă mult, deoarece 4 poate fi asimilat cu 2, în ceea ce privește divizorii. Dacă dorim să reconstituim soluția minim lexicografică, trebuie să adăugăm încă o verificare. În cazul în care pentru o secvență dată, numărul maxim de apariții este deținut de cifrele 3 la egalitate cu numărul de apariții ale cifrelor 2 sau 4, atunci trebuie să verificăm dacă primul element par este 2 sau 4: dacă este 2, vom alege să utilizăm cifrele 2 și 4; dacă primul element este 4, este preferabil să alegem cifrele egale cu 3.
- Dacă secvența conține toate cifrele > 1, observăm că cifra 6 este din nou o cifră specială, fiind o punte de legătură între secvențe de cifre egale cu 3 și secvențe de cifre 2 sau 4. Pentru 5 și 7 vom contoriza numărul de apariții. Pentru 2, 3, 4, vom partiționa din nou secvența în subsecvențe delimitate de cifra 6, vom rezolva problema pentru fiecare subsecvență și vom concatena soluțiile inserând cifrele egale cu 6 pe pozițiile corespunzătoare. La final vom alege, evident, varianta pentru care numărul de apariții este maxim, dar, din nou, la reconstituirea soluției minime din punct de vedere lexicografic trebuie să verificăm dacă numărul maxim de apariții se obține pentru cifra 5 la egalitate cu soluția obținută pentru cifrele 2, 3, 4, 6. Vom verifica dacă prima valoare care apare în soluția cu 2, 3, 4, 6 este egală cu 6 (caz în care va fi preferabil să folosim cifrele 5, în soluția minim lexicografică).

Soluția 2

Rezolvăm problema prin metoda programării dinamice. Formulăm o subproblemă astfel: „să se determine lungimea celui mai lung subșir care începe la o poziție mai mare sau egală cu i și are proprietatea de a fi talisman noros ($1 \leq i \leq N$)".

Vom reține lungimile în vectorul $d[]$, $d[i]$= lungimea celui mai lung subșir care începe la poziția i și are proprietatea de a fi talisman noros ($1 \leq i \leq N$).

În vectorul $next[]$, vom reține informații pentru reconstituirea soluției, $next[i]$ = poziția elementului care urmează după elementul de pe poziția i, în cel mai lung subșir cu proprietatea de a fi talisman norocos minim lexicografic (sau 0 dacă nu există un astfel de element).

Rezolvăm subproblemele în ordinea descrescătoare a dimensiunii acestora, parcurgând vectorul de la N către 1.

Pentru a calcula cel mai lung subșir cu proprietatea de a fi talisman norocos care începe la poziția i, încercăm să adăugăm valoarea $v[i]$ unui subșir maximal de la dreapta acestuia; pentru ca acest lucru să fie posibil, $v[i]$ trebuie să aibă un divizor comun > 1 cu elementul de început al subșirului. Cu alte cuvinte, pentru orice $j > i$, cu $(v[i], v[j])$ având un divizor comun > 1:

$d[i] = max(d[i], d[j] + 1)$;

În caz de egalitate, ținem minte în $next[i]$ cea mai mică valoare următoare pentru care subșirul construit cu aceasta să fie maximal. Vom folosi vectorul $next$ pentru a reconstrui soluția minimă lexicografică la final. După finalizarea dinamicii, este suficient să începem de la poziția i pentru care $d[i]$ e maxim și $v[i]$ e minim, să iterăm după regula $i = next[i]$, și să afișăm cifrele de pe toate pozițiile parcurse.

Soluția descrisă mai sus are complexitatea $O(N^2)$, dar poate fi optimizată folosind următoarea observație: este suficient pentru fiecare cifră de la 0 la 7 să ne uităm doar la cea mai din stânga poziție pe care aceasta apare, pentru a găsi subșirul maximal care începe cu valoarea respectivă.

E ușor de văzut că acest fapt este adevărat, deoarece, pentru două poziții a și b, cu $a < b$ și $v[a] = v[b]$, subșirul maximal care începe în a are lungimea cel puțin egală cu cea a subșirului

maximal care începe în b, plus 1.

Putem construi vectorul auxiliar $last\_pos$:

$last\_pos[c]$ = ultima poziție găsită în iterație a cifrei c.

Astfel, dinamica devine: $d[i] = max(d[i], d[last\_pos[c]]+1)$, pentru fiecare c între 0 și 7 cu $(v[i], c)$ având un divizor comun >1;

La final, $last\_pos[v[i]] = i$.

Acum, în loc să facem N iterații pentru fiecare i, vom face maxim 8. Complexitatea finală este $O(N)$.

9.9 Cod-sursă pentru problema Vnoroc

```
#include <fstream>
using namespace std;
ifstream in("vnoroc.in");
ofstream out("vnoroc.out");

const int maxn = 1000005;
int c,n,v[maxn];

// rezolva in cazul 2-3-4 - folosim solutia intre doi de 6.
// returneaza numarul de elemente ale solutiei optime.
int solve_two_four_three(int st, int dr)
{   int two_count = 0, three_count = 0;
    bool starts_with_four = 0;
    for (int i = st; i <= dr; i++)
    {
        if (v[i] % 2 == 0)
        {
            two_count++;
            if (two_count == 1 && v[i] == 4)
                starts_with_four = 1;
        }
        if (v[i] % 3 == 0) three_count++;
    }
    // eliminam solutia ne-optima; nu eliminam toate elementele totusi,
    // deoarece putem avea valori de 5 sau 7 in intervalul [st,dr]
    if (three_count > two_count || (three_count == two_count && starts_with_four))
    {
        for (int i = st; i <= dr; i++)
        {
            if (v[i] % 2 == 0)
                v[i] = -1;
        }
        return three_count;
    }

    for (int i = st; i <= dr; i++)
    {
        if (v[i] % 3 == 0)
            v[i] = -1;
    }
    return two_count;
}

// rezolva in cazul 1-2-3-4-5-6-7. Folosim solutia intre doi de 0
```

```
// returneaza numarul de elemente ale solutiei optime.
int solve_non_zero(int st, int dr)
{   // vom compara intre solutia care foloseste doar 5,
    // cea care foloseste doar 7,
    // si cea care leaga secvente de 2,4 si 3 cu punti de 6.
    int five_count = 0, seven_count = 0;
    int last_six_position = st - 1;
    int six_solution_size = 0;
    bool starts_with_six = 0;

    for (int i = st; i <= dr; i++)
    {
        if (v[i] == 5) five_count++;
        if (v[i] == 7) seven_count++;
        if (v[i] == 6)
        {
          six_solution_size += 1 + solve_two_four_three(last_six_position+1, i-1);
          last_six_position = i;
          if (six_solution_size == 1) starts_with_six = 1;
        }
    }
    six_solution_size += solve_two_four_three(last_six_position+1, dr);
    if (seven_count > five_count && seven_count > six_solution_size)
    {
        // eliminam elementele care nu ne trebuie in solutie
        for (int i = st; i <= dr; i++)
            if (v[i] != 7)  v[i] = -1;
        return seven_count;
    }
    // alegem 5 in favoarea solutiei cu 2-3-4-6, doar in situatia in care
    // acea solutie incepe cu un 6.
    if (five_count >= seven_count && five_count > six_solution_size ||
            (five_count == six_solution_size && starts_with_six))
    {
        for (int i=st; i<=dr; i++)
            if (v[i]!=5) v[i]=-1;
        return five_count;
    }
    for (int i=st; i<=dr; i++)
        if (v[i]==7||v[i]==5) v[i]=-1;
    return six_solution_size;
}

int main()
{
    in>>c>>n;
    int solution_size=0;
    int last_zero_position=0;

    for (int i=1; i<=n; i++)
    {
        in>>v[i];
        if(v[i]==0)
        {
          solution_size+=1+solve_non_zero(last_zero_position+1, i-1);
          last_zero_position=i;
        }
    }
    solution_size+=solve_non_zero(last_zero_position+1, n);

    if(c==1)
```

```
    {
        out<<n-solution_size<<'\n';
        return 0;
    }
    for (int i=1; i<=n; i++)
        if (v[i]!=1&&v[i]!=-1) out<<v[i]<<' ';
    out<<'\n';
    return 0;
}
```

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
using namespace std;
ifstream f("vnoroc.in");
ofstream g("vnoroc.out");
const int N = 1000010;
vector<int> v[10];
int n,m,p,cer,x[N],nxt[N],lg[10],po[10];
int main()
{
    for(int i=0;i<=9;i++)
        for(int j=9;j>=0;j--)
            if(__gcd(i,j)!=1)
                v[i].push_back(j);
    f>>cer>>n;
    for(int i=1;i<=n;i++)
        f>>x[i];
    for(int i=n;i>=1;i--)
    if(x[i]!=1)
    {
        int a,b;
        a=x[i];b=0;
        for(auto c:v[a])
            if(lg[c]>=lg[b])
                b=c;
        nxt[i]=lg[b]?po[b]:n+1;
        lg[a]=lg[b]?lg[b]+1:1;
        po[a]=i;
    }
    for(int i=0;i<=9;i++)
        m=max(m,lg[i]);
    if(cer==1)
    {
        g<<n-m<<'\n';
        return 0;
    }
    for(int i=9;i>=0;i--)if(lg[i]==m)p=po[i];
    for(;p<=n;p=nxt[p])g<<x[p]<<' ';
    g<<'\n';
    return 0;
}
```

Partea a III-a

Tabăra de pregătire a lotului național de informatică juniori

Craiova, 9-14 mai 2025

Capitolul 10

Barajul 1

10.1 Problema Rețete

Propusă de: prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași
stud. Andrei Boacă, Facultatea de Informatică, Universitatea „Al. I. Cuza” Iași

De când am descoperit Chat GPT îl folosesc la orice, inclusiv în bucătărie. De exemplu, astăzi am făcut un inventar al ingredientelor pe care le am în casă sub forma unei liste, în care fiecare linie corespunde unui ingredient sub forma:

```
denumire_ingredient cantitate unitate_de_măsură
```

Unitatea de măsură poate fi litrul (indicat prin `l`), decilitrul (indicat prin `dl`; `1l=10dl`), centilitrul (indicat prin `cl`; `1l=100cl`) mililitrul (indicat prin `ml`; `1l=1000ml`), gramul (indicat prin `g`), kilogramul (indicat prin `kg`; `1kg=1000g`) sau bucata (indicată prin `b`). Cantitatea și denumirea ingredientului, respectiv cantitatea și unitatea de măsură sunt separate prin câte un spațiu.

Apoi i-am dat lui Chat GPT rețetele mele de prăjituri preferate, fiecare rețetă în următorul format:

```
#nr
descriere listă ingrediente
```

Rețetele sunt numerotate începând cu 1, în ordinea în care sunt scrise (numărul rețetei fiind specificat după caracterul `#`). Descrierea listei de ingrediente este în același format cu inventarul ingredientelor pe care le am în casă. Nefiind interesant pentru Chat GPT, am omis descrierea modului de preparare. I-am cerut lui Chat GPT să-mi răspundă la două întrebări:

1. Dacă aș dori să prepar prăjitura dintr-o singură rețetă, care sunt rețetele de prăjituri pe care le-aș putea prepara cu ingredientele pe care le am în casă?
2. Dacă aș dori să prepar mai multe prăjituri, care este numărul maxim de rețete de prăjituri pe care le-aș putea prepara cu ingredientele din inventar, precum și toate combinațiile cu număr maxim de rețete de prăjituri ce aș putea să le prepar.

Chat GPT nu s-a prea descurcat, prin urmare scrieți voi un program care să răspundă la întrebări.

Cerințe

Date fiind lista cu inventarul ingredientelor pe care le am în casă, precum și rețetele de prăjituri:

1. determinați numerele de ordine ale rețetelor ce pot fi preparate cu ingredientele din inventar;

2. determinați numărul maxim de rețete distincte de prăjituri care pot fi preparate cu ingredientele din inventar, precum și toate combinațiile cu număr maxim de rețete.

Date de intrare

Fișierul de intrare `retete.in` conține pe prima linie numărul natural C reprezentând cerința care trebuie să fie rezolvată (1 sau 2). Pe următoarele linii este scris inventarul ingredientelor, câte un ingredient pe o linie, în formatul descris în enunț. În continuare, până la sfârșitul fișierului sunt scrise rețetele, în formatul din enunț. Pe ultima linie a fișierului de intrare se află caracterul `*`.

Date de ieșire

Dacă cerința $C = 1$, fișierul de ieșire `retete.out` conține o singură linie, pe care sunt scrise în ordine crescătoare, separate prin câte un spațiu, numerele rețetelor care pot fi preparate cu ingredientele din inventar. Dacă cerința $C = 2$, fișierul de ieșire `retete.out` conține pe prima linie numărul maxim de rețete care pot fi preparate cu ingredientele din inventar. Pe următoarele linii sunt scrise în ordine lexicografică combinațiile cu număr maxim de rețete care pot fi preparate cu ingredientele din inventar, fiecare combinație pe o linie. O combinație este o succesiune de numere de rețete, în ordine strict crescătoare, separate prin câte un spațiu.

Restricții

- În inventar există cel mult 10000 de ingrediente **distincte**.
- În fișierul de intrare există cel mult 100 de rețete, fiecare rețetă având cel mult 100 de ingrediente, **nu neapărat distincte**.
- Denumirea unui ingredient are cel mult 30 de caractere (litere mici, cifre, spațiu sau procent).
- Cantitățile sunt numere naturale nenule ≤ 10000.
- Dacă $C = 2$, cu ingredientele din inventar pot fi preparate, individual, cel mult 18 rețete.
- Un ingredient poate avea unitatea de măsură doar din una dintre următoarele 3 categorii: unitate de masă (`kg`, `g`), unitate de volum (`l`, `dl`, `cl`, `ml`) sau bucata (`b`).
- Combinația de rețete a_1, a_2, $\ldots a_N$ precedă în ordine lexicografică combinația de rețete b_1, b_2, $\ldots b_N$ dacă există k ($1 \leq k \leq N$), astfel încât $a_i = b_i$, pentru orice $1 \leq i < k$ și $a_k < b_k$.

#	Puncte	Restricții
1	30	$C = 1$
2	70	$C = 2$

Exemple

retete.in	retete.out
1 unt 82% 500 g ulei de floarea soarelui 5 l faina 000 4 kg smantana 500 g zahar 1500 g zahar pudra 200 g rahat 500 g cacao 450 g rom 2 dl nuca 2 kg unt 60% 800 g vanilie 10 g zahar vanilat 5 b ou 30 b #1 nuca 100 g ou 2 b zahar pudra 100 g unt 82% 220 g faina 000 300 g ou 1 b zahar vanilat 1 b vanilie 1 g #2 biscuit 500 g unt 60% 200 g rahat 150 g nuca 150 g zahar 100 g cacao 60 g lapte 200 ml #3 unt 82% 250 g smantana 250 g faina 000 750 g rahat 400 g #4 unt 82% 100 g faina 000 300 g zahar 100 g *	1 3 4

2 unt 82% 500 g ulei de floarea soarelui 5 l faina 000 4 kg smantana 500 g zahar 1500 g zahar pudra 200 g rahat 500 g cacao 450 g rom 2 dl nuca 2 kg unt 60% 800 g zahar vanilat 5 b ou 30 b #1 nuca 100 g ou 2 b zahar pudra 100 g unt 82% 220 g faina 000 300 g ou 1 b zahar vanilat 1 b #2 biscuit 500 g unt 60% 200 g rahat 150 g nuca 150 g zahar 100 g cacao 60 g lapte 200 ml #3 unt 82% 250 g smantana 250 g faina 000 750 g rahat 400 g #4 unt 82% 100 g faina 000 300 g zahar 100 g *	2 1 3 1 4 3 4

Explicații

Doar rețetele 1, 3 și 4 pot fi preparate cu ingredientele din inventar (deoarece pentru rețeta 2 este necesar să avem lapte și biscuiți, care nu există în inventar).
Maximum 2 rețete am putea prepara cu ingredientele din inventar, acestea fiind 1 și 3, sau 1 și 4 sau 3 și 4.

10.2 Rezolvarea problemei Rețete

Vom citi linie cu linie ingredientele din inventar. Pentru a extrage ingredientele, o soluție simplă este de a parcurge șirul de la final spre început:

- căutăm ultimul spațiu (acesta delimitează unitatea de măsură); pentru a nu avea probleme cu cantități exprimate în unități de măsură diferite, convertim l, dl, cl în ml, iar kg în g;
- căutăm următorul spațiu, acesta delimitează cantitatea;
- restul șirului reprezintă denumirea produsului.

Ingredientele din lista-inventar le vom memora într-o structură de date, împreună cu cantitățile în care sunt disponibile. Structura de date pe care o utilizați pentru memorarea ingredientelor influențează eficiența implementării.

Cerința 1

Vom citi succesiv rețetele și pentru fiecare rețetă verificăm dacă toate ingredientele specificate în rețeta respectivă apar în inventar într-o cantitate suficient de mare.

Cerința 2

Vom nota cu P numărul de rețete care pot fi preparate individual. Pentru cerința 2 se garantează că există cel mult 18 rețete care pot fi preparate individual ($P \leq 18$). Prin urmare, vor exista cel mult $C_{18}^{9} = 48\,620$ combinații fezabile.

Această observație ne sugerează că este o simplă problemă de generare de elemente combinatoriale, care poate fi abordată în diferite moduri. În funcție de modul de abordare și de implementare pot fi obținute diferite punctaje.

Soluția 1. Generare submulțimi

Se generează toate submulțimile de rețete și pentru fiecare submulțime se verifică dacă este fezabilă (adică dacă toate ingredientele care apar în rețetele respective există în inventar, iar suma cantităților necesare pentru fiecare ingredient este mai mică sau egală cu cantitatea disponibilă).

Această soluție are complexitate $O(2^P \cdot P \cdot I)$, unde I este numărul maxim de ingrediente din fiecare rețetă ($I \leq 100$), datorită faptului că pentru fiecare submulțime generată trebuie să verificăm dacă suma cantităților ingredientelor care apar în submulțime este disponibilă.

Soluția 2. Generare combinări

Se generează combinări (submulțimi de K rețete dintre cele P care pot fi preparate), în ordine descrescătoare după K) și se verifică, în același mod, pentru fiecare combinare dacă este fezabilă. La primul K pentru care există soluții fezabile ne oprim, întrucât acestea sunt combinații cu număr maxim de rețete. Această abordare este mai eficientă decât prima, dar nu obține 100 de puncte.

Soluția 3. Generare cod Gray

Pentru început vom „normaliza” șirurile de caractere, asociind fiecăruia câte un număr, pentru a putea scăpa de un eventual factor de log în implementarea ideii de mai jos. Normalizarea o vom face folosind un `std::map<std::string , int>` asociind fiecărui șir de caractere (ținut în memorie pe tipul de date `std::string`) neîntâlnit până la momentul curent un nou număr.

Vom genera submulțimile de rețete sub formă de măști pe biți în ordinea codului Gray. Codul $Gray(K)$ (codul $Gray$ pe K biți) se obține în felul următor:

1. $Gray(1)$ este 0,1.
2. $Gray(K)$ pentru $K > 1$ se obține din $Gray(K-1)$ completând codul $Gray(K-1)$ cu un bit cu valoarea 0, apoi concatenând cele 2^{K-1} soluții cu oglindirea codului $Gray(K–1)$ completată cu un bit cu valoarea 1.

De exemplu, $Gray(2)$ poate fi:
00
01
11
10

$Gray(3)$ poate fi:
000
001
011
010
110
111
101
100

Codul Gray poate fi calculat și direct cu formula `x ^ (x >> 1)`.

Avantajul folosirii acestui mod de generare a submulțimilor este dat de faptul că valorile de la două poziții vecine diferă prin exact un bit, motiv pentru care va trebui să actualizăm suma cantităților ingredientelor din rețetele submulțimii curente doar pentru rețeta care apare/dispare din submulțimea de la pasul anterior. Obținem astfel o complexitate $O(2^P \cdot I)$, care este suficientă pentru punctajul maxim.

Soluția 4. Backtracking optimizat

O abordare de tip backtracking optimizat pentru generarea soluțiilor poate obține, de asemenea, punctaj maxim.

10.3 Cod-sursă pentru problema Rețete

```
#include <fstream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
ofstream fout("retete.out");
int cerinta;
struct date
{
    int nume;
    int c;
};
vector<string> nrm;
int nrcrt;
int f[50005], f2[50005];
vector<date> reteta[105];
int getnr(string x, string y)
{   int nr = 0;
    for (char c : x) nr = nr * 10 + c - '0';
    if (y == "l" || y == "kg") nr = nr * 1000;
    if (y == "dl")  nr = nr * 100;
    if (y == "cl")  nr = nr * 10;
    return nr;
}
int getnorm(string s)
```

```
{
    int st = 0, dr = nrm.size();
    dr--;
    while (st <= dr)
    {
        int mij = (st + dr) / 2;
        if (nrm[mij] == s)
            return mij + 1;
        if (nrm[mij] < s)
            st = mij + 1;
        else
            dr = mij - 1;
    }
    return -1;
}
date parse(string s)
{
    string unit, num;
    while (s.back() != ' ')
    {
        unit.push_back(s.back());
        s.pop_back();
    }
    s.pop_back();
    while (s.back() != ' ')
    {
        num.push_back(s.back());
        s.pop_back();
    }
    s.pop_back();
    reverse(unit.begin(), unit.end());
    reverse(num.begin(), num.end());
    int nr = getnr(num, unit);
    return { getnorm(s),nr };
}
void initparse(string s)
{
    string unit, num;
    while (s.back() != ' ')
    {
        unit.push_back(s.back());
        s.pop_back();
    }
    s.pop_back();
    while (s.back() != ' ')
    {
        num.push_back(s.back());
        s.pop_back();
    }
    s.pop_back();
    reverse(unit.begin(), unit.end());
    reverse(num.begin(), num.end());
    nrm.push_back(s);
}

vector<int> good;
int lgmax, lg;
vector<int> me;
vector<vector<int>> sol;

vector<int> gray(int lg)
```

```
{
    if (lg == 1) return { 0,1 };
    vector<int> rez = gray(lg - 1);
    vector<int> aux = rez;
    for (int i = 0;i < aux.size();i++)
        aux[i] += (1 << (lg - 1));
    reverse(aux.begin(), aux.end());
    for (int i : aux)  rez.push_back(i);
    return rez;
}
bool bysize(int a, int b)
{
    return reteta[a].size() < reteta[b].size();
}
string ss;
ifstream fin2("retete.in");
int main()
{
    fin2 >> cerinta; fin2.get();
    while (true)
    {
        getline(fin2, ss);
        if (ss[0] == '*') break;
        if (ss[0] != '#' && ss[0] != '*')
            initparse(ss);
    }
    fin2.close();
    ifstream fin("retete.in");
    sort(nrm.begin(), nrm.end());
    vector<string> aux;
    for (string ss : nrm)
        if (aux.empty() || aux.back() != ss)
            aux.push_back(ss);
    nrm = aux;
    fin >> cerinta; fin.get();
    string curent;
    while (true)
    {
        getline(fin, curent);
        if (curent[0] == '#') break;
        else
        {
            date x = parse(curent);
            f[x.nume] += x.c;
        }
    }
    int ind = 1;
    vector<int> names;
    while (true)
    {
        getline(fin, curent);
        if (curent[0] == '#' || curent[0] == '*')
        {
            vector<date> v;
            for (int i : names)
            {
                v.push_back({ i,f2[i] });
                f2[i] = 0;
            }
            names.clear();
            reteta[ind] = v;
```

```
            if (curent[0] == '*') break;
            ind++;
            continue;
        }
        date x = parse(curent);
        if (f2[x.nume] == 0)
            names.push_back(x.nume);
        f2[x.nume] += x.c;
    }
    for (int i = 1;i <= ind;i++)
    {
        bool ok = 1;
        for (date x : reteta[i])
            ok &= (f[x.nume] >= x.c);
        if (ok)
            good.push_back(i);
    }
    if (cerinta == 1)
    {
        for (int i : good) fout << i << ' ';
        return 0;
    }
    lg = good.size();
    sort(good.begin(), good.end(), bysize);
    int lgmax = 0;
    vector<int> masks = gray(lg);
    int negative = 0;
    for (int z = 1;z < masks.size();z++)
    {
        vector<int> vec;
        int mask = masks[z];
        int prv = masks[z - 1];
        int where = (mask ^ prv);
        int semn = 1;
        if (mask < prv)
            semn = -1;
        for (int bit = 0;bit < lg;bit++)
        {
            if ((where >> bit) & 1)
                for (date me : reteta[good[bit]])
                {
                    if (f[me.nume] < 0)
                        negative--;
                    f[me.nume] -= me.c * semn;
                    if (f[me.nume] < 0)
                        negative++;
                }
            if ((mask >> bit) & 1)
                vec.push_back(good[bit]);
        }
        if (negative > 0) continue;
        if (vec.size() > lgmax)
        {
            lgmax = vec.size();
            sol.clear();
            sol.push_back(vec);
        }
        else if (vec.size() == lgmax)
            sol.push_back(vec);
    }
    for (int i = 0;i < sol.size();i++)
```

```
        sort(sol[i].begin(), sol[i].end());
    sort(sol.begin(), sol.end());
    fout << lgmax << '\n';
    for (auto i : sol)
    {
        for (auto j : i) fout << j << ' ';
        fout << '\n';
    }
    return 0;
}
```

```
/* backtracking optimizat*/
#include <bits/stdc++.h>
#pragma GCC optimize ("Ofast")
#pragma GCC optimize ("unroll-loops")
#pragma GCC target ("avx2")
using namespace std;
ifstream fin("retete.in");
ofstream fout("retete.out");
mt19937 rng(chrono::steady_clock().now().time_since_epoch().count());
int cerinta;
struct date
{
    int nume;
    int c;
};
map<string,int> nrm;
int nrcrt;
int f[50005],f2[50005];
vector<date> reteta[105];
int getnr(string x,string y)
{
    int nr=0;
    for(char c:x)
        nr=nr*10+c-'0';
    if(y=="l"||y=="kg")
        nr=nr*1000;
    if(y=="dl")
        nr=nr*100;
    if(y=="cl")
        nr=nr*10;
    return nr;
}
int getnorm(string s)
{
    if(nrm.count(s)==0)
        nrm[s]=++nrcrt;
    return nrm[s];
}
date parse(string s)
{
    string unit,num;
    while(s.back()!=' ')
    {
        unit.push_back(s.back());
        s.pop_back();
    }
    s.pop_back();
    while(s.back()!=' ')
    {
        num.push_back(s.back());
        s.pop_back();
```

```
    }
    s.pop_back();
    reverse(unit.begin(),unit.end());
    reverse(num.begin(),num.end());
    int nr=getnr(num,unit);
    return {getnorm(s),nr};
}
vector<int> good;
int lgmax,lg;
vector<int> me;
vector<vector<int>> sol;
void bkt(int mask,int bit)
{
    if(bit==lg)
    {
        vector<int> aux=me;
        if(aux.size()>lgmax)
        {
            sort(aux.begin(),aux.end());
            lgmax=aux.size();
            sol.clear();
            sol.push_back(aux);
        }
        else if(aux.size()==lgmax)
        {
            sort(aux.begin(),aux.end());
            sol.push_back(aux);
        }
        return;
    }
    bkt(mask,bit+1);
    bool ok=1;
    for(date x:reteta[good[bit]])
    {
        f[x.nume]-=x.c;
        if(f[x.nume]<0)
            ok=0;
    }
    me.push_back(good[bit]);
    if(ok)
        bkt(mask+(1<<bit),bit+1);
    for(date x:reteta[good[bit]])
        f[x.nume]+=x.c;
    me.pop_back();
}

bool bylg(int a,int b)
{
    return reteta[a].size()>reteta[b].size();
}

int main()
{
    ios_base::sync_with_stdio(false); fin.tie(0);
    fin>>cerinta;  fin.get();
    string curent;
    while(true)
    {
        getline(fin,curent);
        if(curent[0]=='#')
            break;
```

```
        else
        {
            date x=parse(curent);
            f[x.nume]+=x.c;
        }
    }
    int ind=1;
    while(true)
    {
        getline(fin,curent);
        if(curent[0]=='#'||curent[0]=='*')
        {
            vector<date> v;
            for(int i=1;i<=nrcrt;i++)
                if(f2[i]!=0)
                    v.push_back({i,f2[i]});
            reteta[ind]=v;
            for(int i=1;i<=nrcrt;i++)
                f2[i]=0;
            if(curent[0]=='*')
                break;
            ind++;
            continue;
        }
        date x=parse(curent);
        f2[x.nume]+=x.c;
    }
    for (int i=1;i<=ind;i++)
    {
        bool ok=1;
        for (date x:reteta[i])
            ok&=(f[x.nume]>=x.c);
        if (ok) good.push_back(i);
    }
    if(cerinta==1)
    {
        for(int i:good) fout<<i<<' ';
        return 0;
    }
    lg=good.size();
    sort(good.begin(),good.end(),bylg);
    bkt(0,0);
    sort(sol.begin(),sol.end());
    fout<<lgmax<<'\n';
    for(auto i:sol)
    {
        for(auto j:i) fout<<j<<' ';
        fout<<'\n';
    }
    return 0;
}
```

10.4 Problema Tort

Propusă de: instr. Cristian Frâncu, Nerdvana București

Construim un tort. Pornim cu o foaie de grosime A. Putem efectua una din două operații posibile:

- Împăturim tortul, îndoind foaia pe din două. Grosimea tortului se dublează.
- Adăugăm un strat de unt pe deasupra. Stratul de unt are grosime B. Grosimea tortului crește cu B.

Cerințe

1. Dându-se A, B și C să se spună care este grosimea minimă a unui tort ce are grosime cel puțin egală cu C.
2. Dându-se A, B și C să se spună care este numărul minim de operații în care se poate obține acea grosime minimă.

Date de intrare

Pe prima linie a fișierului de intrare `tort.in` se vor găsi numărul cerinței, P, și numărul de teste, T. Pe următoarele T linii se vor găsi triplete A B C.

Date de ieșire

În fișierul de ieșire `tort.out` afișați răspunsul cerut pentru fiecare triplet de la intrare, câte unul pe linie.

Restricții

- $1 \leq A, B \leq 10^9$
- $\max(A, B) \leq C \leq 10^{17}$
- $0 \leq T \leq 10^6$

#	Puncte	Restricții
1	6	$P = 1$, $A = B$
2	7	$P = 2$, $A = 1$, $B = 1$ și C are forma $2^k - 1$
3	10	$P = 2$, $A = 1$, $B = 1$
4	15	$P = 1$, $T = 1$, $500 \leq A, B \leq 1\,000$, $10\,000 \leq C \leq 200\,000$
5	7	$P = 2$, $T = 1$, $500 \leq A, B \leq 1\,000$, $10\,000 \leq C \leq 200\,000$
6	19	$P = 1$, fără restricții suplimentare
7	36	$P = 2$, fără restricții suplimentare

Exemple

tort.in	tort.out	Explicații
1 2 90 100 480 15 25 507	480 510	$480 = (90 + 100) \times 2 + 100$ $510 = [(15 + 25 + 25 + 25 + 25) \times 2 + 25] \times 2$

2 2 90 100 480 15 25 507	3 7	
1 1 10 10 632	640	$640 = 10 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$ sau $640 = (10 + 10) \times 2 \times 2 \times 2 \times 2 \times 2$
2 1 10 10 632	6	
2 1 3 7 767	8	Tortul va avea grosime minimă 768, obținută astfel: $768 = 3 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$

10.5 Rezolvarea problemei Tort

Observație: Orice grosime $2^n \times a + m \times b$ poate fi obținută prin îndoiri și ungeri cu unt, deoarece putem să efectuăm n îndoiri și apoi m ungeri.

Cerința 1

- Vom încerca toate variantele pentru n, deoarece 2^n va depăși rapid c.
- Pentru un n dat calculăm m minim astfel încât $2^n \times a + m \times b \geq c$.
- Astfel:

  $$m = (c - 2^n \times a)/b$$

  dacă împărțirea este exactă sau

  $$m = (c - 2^n \times a)/b + 1$$

  dacă împărțirea este cu rest.
- Astfel, formula finală a lui m este

  $$m = (c - 2^n \times a + b - 1)/b$$
- Grosimea tortului va fi:

  $$g = 2^n \times a + m \times b$$

  Înlocuind:

  $$g = 2^n \times a + (c - 2^n \times a + b - 1)/b \times b$$

Algoritmul pentru determinarea grosimii minime a tortului este:

---

citește a, b, c
grosime_minima $\leftarrow \infty$
$n \leftarrow 0$
while $2^n \times a \leq c$ **do**
    $g \leftarrow 2^n \times a + (c - 2^n \times a + b - 1)/b \times b$
    if $g <$ grosime_minima **then**
        grosime_minima $\leftarrow g$
    $n \leftarrow n + 1$
afișează grosime_minima

Complexitatea este $O(\log c)$ ca timp și $O(1)$ memorie.

Cerința 2

Observație: O combinație $2^n \times a + m \times b$ se obține în număr minim de operații astfel:

- Avem n îndoiri.
- Vrem să obținem cele m straturi de unt din cât mai puține aplicări.
- Pentru aceasta trebuie să „strecurăm" puterile lui 2 din m printre îndoiri.
- Este posibil ca n să fie prea mic, caz în care vom adăuga mai multe puteri ale lui 2 din m la început.

Cu alte cuvinte, pentru o combinație $2^n \times a + m \times b$ vom obține numărul minim de operații astfel:

- La început vom unge k straturi de unt, unde $k = m/2^n$.
- Apoi la fiecare dublare vom unge un singur strat doar dacă puterea corespunzătoare a lui 2 din m există în m (are bit 1 în reprezentarea binară a lui m).
- Informatic spus, numărul minim de mutări este:

$$n + m/2^n + \text{popcount}(m \bmod 2^n)$$

  unde `popcount(x)` este numărul de biți 1 din reprezentarea binară a lui x.

Algoritmul pentru determinarea numărului minim de operații este:

---

citește a, b, c
grosime_minima $\leftarrow \infty$
$n \leftarrow 0$
while $2^n \times a \leq c$ **do**
    $m \leftarrow (c - 2^n \times a + b - 1)/b$
    $g \leftarrow 2^n \times a + m \times b$
    if $g <$ grosime_minima **then**
        grosime_minima $\leftarrow g$
        nrop_minim $\leftarrow n + m/2^n + \text{popcount}(m \bmod 2^n)$
    else if $g =$ grosime_minima **then**
        operatii $= n + m/2^n + \text{popcount}(m \bmod 2^n)$
        if operatii $<$ nrop_minim **then**
            nrop_minim $\leftarrow$ operatii
    $n \leftarrow n + 1$
afișează nrop_minim

Complexitatea este $O(\log c \times p)$ ca timp și O(1) memorie, unde p este timpul de calcul al funcției `popcount(x)`. Cu o metodă brută p este $O(\log x)$, dar există și metode mai rapide ce duc la timp $O(\log \log x)$.

Observăm că putem folosi această implementare pentru a rezolva și prima cerință.

Anexă

Algoritmul de mai sus, implementat eficient, se va încadra în timp, dar la limită, existând riscul să depășim timpul cu o implementare mai puțin eficientă „de concurs". Ce putem face?

Să observăm că fișierul de intrare va fi foarte mare, trei milioane de numere. Precum știm (sau nu :-) citirea cu *streams* din C++ sau cu `fscanf` în C este destul de lentă. Putem citi mai rapid

aceste numere folosind o combinație de funcție de bibliotecă `fread()` și prelucrare a caracterelor, numită „citire rapidă” sau „parsing”, în limbajul olimpicilor.

Iată mai jos un exemplu de cod ce citește rapid un întreg `unsigned long long`.

```
#define BUFSIZE (128 * 1024)

FILE *fin, *fout;
int rpos = BUFSIZE - 1; char rbuf[BUFSIZE];

static inline char readChar() {
  if ( !(rpos = (rpos + 1) & (BUFSIZE - 1)) )
    fread( rbuf, 1, BUFSIZE, fin );
  return rbuf[rpos];
}

unsigned long long readInt() {
  int ch;
  unsigned long long res = 0;

  while ( isspace( ch = readChar() ) );
  do
    res = 10 * res + ch - '0';
  while ( isdigit( ch = readChar() ) );

  return res;
}
```

Această citire poate de fi de două până la patru ori mai rapidă decât citirea standard.

10.6 Cod-sursă pentru problema Tort

```
#include <stdio.h>
#include <bits/stdc++.h>

static inline unsigned long long findOp( int n, unsigned long long m ) {
  return n + __builtin_popcountll(m & ((1ULL << n) - 1)) + (m >> n);
}

int main( int argc, char *argv[] ) {
  FILE *fin, *fout;
  int t, cer, i, n;
  unsigned long long p, a, b, c, thick, minthick, m, op, minop;

  fin = fopen( "tort.in", "r" );
  fout = fopen( "tort.out", "w" );
  fscanf( fin, "%d%d", &cer, &t );

  for ( i = 0; i < t; i++ ) {
    fscanf( fin, "%lld%lld%lld", &a, &b, &c );

    // incercam toate variantele 2^n * a + m * b
    n = 0; // pornim cu perechea 2^0 * a + (c - a) / b * b
    minthick = a + (minop = m = ((c - a + b - 1) / b)) * b;
    p = 2 * a;
    n++;
    while ( p <= c ) { // cata vreme 2^n * a nu depaseste c
      thick = p + (m = ((c - p + b - 1) / b)) * b;
      if ( thick < minthick ) {
        minthick = thick;
```

```
        minop = findOp( n, m );
      } else if ( thick == minthick && (op = findOp( n, m )) < minop )
        minop = op;

      p *= 2; // trecem la urmatorul 2^n * a
      n++;
    }

    // testam si ultimul 2^n * a care este strict mai mare decat c
    // in acest caz avem zero ungeri cu unt (m = 0) si nr op este n
    if ( p < minthick ) {
      minthick = p;
      minop = n;
    } else if ( p == minthick && n < minop )
      minop = n;

    fprintf( fout, "%llu\n", cer == 1 ? minthick : minop );
  }

  fclose( fin );
  fclose( fout );

  return 0;
}
```

10.7 Problema Zid

Propusă de: prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian, Drobeta Turnu Severin

Un monument istoric are forma unui zid circular format din N turnuri. Fiecare turn este construit din cărămizi zidite unele peste altele. Înălțimea unui turn este egală cu numărul de cărămizi din care este format turnul.

Zidul trebuie renovat astfel încât, după renovare, turnurile din zid să aibă aceeași înălțime. Înălțimea finală a zidului renovat trebuie să fie cât mai mică.

Pentru renovare se va utiliza o mașină care, la o comandă, alege două turnuri vecine și adaugă în cele două turnuri alese același număr de cărămizi.

În situația în care problema nu are soluție, vor trebui eliminate din zid un număr minim de cărămizi, astfel încât, după eliminare, renovarea să fie posibilă.

Cerințe

Dacă problema nu are soluție, determinați **nrmin**, numărul minim de cărămizi care trebuie eliminate astfel încât, după eliminare, renovarea să poată avea loc.

Dacă problema are soluție, determinați **hmin**, înălțimea finală minimă după renovare.

Date de intrare

Fișierul de intrare `zid.in` conține pe prima linie numărul N. Pe a doua linie sunt scrise N numere naturale $h_1\ h_2 \ldots h_N$ separate prin câte un spațiu, reprezentând înălțimile inițiale ale turnurilor, în ordine de la 1 la N.

Date de ieșire

În fișierul de ieșire `zid.out` se va scrie pe primul rând unul dintre numerele **nrmin** sau **hmin**, după caz.

Restricții

- $1 \leq N \leq 100\,000$
- $0 \leq h_k \leq 1\,000\,000$, $1 \leq k \leq N$
- Se garantează că răspunsul este cel mult egal cu 10^9.

#	Puncte	Restricții
1	9	Există două înălțimi egale cu 1 și $N-2$ înălțimi egale cu 0.
2	12	$h_1 \leq h_2 \leq \cdots \leq h_N$; problema are soluție
3	18	$N \leq 2\,000$; problema are soluție și $hmin \leq 1000$
4	21	$2000 < N \leq 5\,000$; problema are soluție și $1000 < hmin \leq 3000$
5	40	Fără restricții suplimentare.

Exemple

zid.in	zid.out
4 1 2 4 3	4
2 1 3	2
5 5 2 1 3 4	5

Explicații

Pentru exemplul 1, se pot aplica următoarele comenzi:

- la pozițiile 1 și 2 se zidesc câte 2 cărămizi și se obține zidul 3 4 4 3;
- la pozițiile 1 și 4 se zidește câte o cărămidă și se obține zidul 4 4 4 4.

Exemplul 2 nu se poate rezolva decât dacă se vor elimina 2 cărămizi din turnul 2.

Pentru exemplul 3, se pot aplica următoarele comenzi:

- la pozițiile 3 și 4 se zidește câte o cărămidă și se obține zidul 5 2 2 4 4;
- la pozițiile 2 și 3 se zidesc câte trei cărămizi și se obține zidul 5 5 5 4 4;
- la pozițiile 4 și 5 se zidește câte o cărămidă și se obține zidul 5 5 5 5 5.

10.8 Rezolvarea problemei Zid

Multe rezolvări pentru cerința 1 pornesc de la următoarea observație. Oricând avem $h_i > h_{i+1}$, cumva trebuie să-l aducem pe h_{i+1} la nivelul lui h_i (cel puțin). Nu ajută să creștem perechea (h_i, h_{i+1}), căci diferența dintre ele se va păstra. De aceea, este necesar să creștem perechea (h_{i+1}, h_{i+2}) cu valoarea $h_i - h_{i+1}$.

Aceste creșteri sunt strict necesare: nu putem să egalizăm h_i și h_{i+1} fără ele. Deci, dacă problema are soluție, orice algoritm care aplică astfel de creșteri va ajunge la soluție.

Subtaskul 1

Fie p și q, $p < q$, pozițiile celor două înălțimi 1 și fie $A = [p+1, q-1]$ și $B = [q+1, p-1]$ intervalele (circulare) dintre ele. Apar trei posibilități.

1. Dacă N este impar, atunci exact unul dintre A și B are lungime pară, să spunem B. Atunci pe B îl aducem la înălțimea 1 cu creșteri din două în două poziții. Pe A îl umplem similar și va rămâne o înălțime 0, să zicem pe poziția $p+1$. Mai creștem o dată perechea $(p, p+1)$ și obținem un zid cu o înălțime 2 și restul 1. Numărul de înălțimi 1 este par, deci putem completa zidul la înălțime 2.
2. Dacă N este par, iar A și B au lungimi pare, atunci putem completa zidul la înălțime 1.
3. Dacă N este par, iar A și B au lungimi impare, atunci putem crește perechi din A și din B până rămân doar două înălțimi de 0, să zicem la pozițiile $p+1$ și $q+1$. Dacă acum creștem perechile $(p, p+1)$ și $(q, q+1)$, obținem un zid cu înălțimi 2 pe pozițiile p și q și 1 în rest. Adică am ajuns la problema originală crescută cu o cărămidă pe toate pozițiile. Dar poate există alte creșteri care duc la o soluție? Răspunsul este că nu, deoarece orice creștere are loc pe o poziție pară și o poziție impară. Suma pe pozițiile de aceeași paritate cu p și q va fi mereu cu 2 mai mare decât suma pe pozițiile de paritate opusă. Problema nu are soluție decât dacă eliminăm cei doi de 1 inițiali.

Reținem de aici observația-cheie că, dacă N este par, atunci problema are soluție doar dacă suma pe pozițiile pare este egală cu suma pe pozițiile impare.

Subtaskul 2

Dacă zidul este nedescrescător, iar problema are soluție, atunci singurul mod în care N poate fi par este că perechile de poziții 1-2, 3-4, 5-6 etc. au înălțimi egale. Deci putem crește aceste perechi pentru a aduce zidul la înălțimea h_N.

Dacă N este impar, putem parcurge zidul de la N la 1. Creștem fiecare înălțime h_i la valoarea h_N, crescând corespunzător și poziția h_{i-1}. La final, când creștem h_1 la înălțimea h_N, h_N va crește și el până la o nouă înălțime H. Acum, avem o poziție de înălțime H și un număr par $(N-1)$ de poziții de înălțime h_1, pe care le putem crește în perechi pînă la H. Așadar, răspunsul este H, iar complexitatea este $O(N)$.

Subtaskurile 3 și 4

Putem aplica același principiu și la un vector de formă oarecare. În mod repetat, căutăm minimul y. Fie x și z vecinii săi cu $x \geq z$. Atunci îl aducem pe y la înălțimea lui x, crescând perechea (y, z) cu valoarea $x - y$.

Remarcăm că aceste soluții fac efort proporțional cu înălțimea finală a zidului. Această înălțime poate fi $N/2$, de exemplu pentru vectorul $1010 \ldots 101$.

Pentru subtaskul 3 este suficientă o implementare în $O(\log N)$ per creștere. Putem menține un `std::set` cu coloanele ordonate după înălțime. La înălțimi egale preferăm coloana cu vecinul cel mai înalt, ca să ne asigurăm că nu alegem o coloană de mijloc dintr-un platou de valori egale. Este nevoie de atenție la implementare, întrucât, la creșterea unei perechi (a, b), atât elementele, cât și ceilalți doi vecini ai lor își schimbă criteriul de ordonare, deci toate patru trebuie șterse și reinserate în structură. Complexitatea este $O(\text{hmin} \cdot N \cdot \log N)$.

Putem îmbunătăți constanta acestei implementări, păstrând complexitatea asimptotică, dacă grupăm în permanență platourile de coloane egale. Pentru a aduce un triplet x_1, x_2, x_3 cu $x_1 \geq x_2 \leq x_3$ la aceeași înălțime vom face următoarele operații:

- Aducem x_1 și x_3 la aceeași înălțime prin creșterea celui mai mic dintre ele, apelând o operație de incrementare care să cuprindă atât elementul mai mic, cât și x_2.
- Alternăm între operația (x_1, x_2) și operația (x_2, x_3) pentru a aduce x_2 la nivel cu x_1 și x_3.

Acum în loc de trei coloane egale putem păstra una singură, deoarece pe celelalte două le vom crește împreună.

Pentru a trece și subtaskul 4, putem reduce timpul de găsire a minimului la $O(1)$ dacă menținem pentru fiecare înălțime o listă înlănțuită cu coloanele de acea înălțime. Complexitatea este $O(\text{hmin} \cdot N)$.

Subtaskul 5

Notăm cu x_k valoarea care se adaugă la turnurile h_k și h_{k+1}. Deoarece toate turnurile vor avea aceeași înălțime vom avea:

$$
\begin{aligned}
& h_1 + x_N + x_1 \\
&= h_2 + x_1 + x_2 \\
&= h_3 + x_2 + x_3 = \ldots \\
&= h_{N-1} + x_{N-2} + x_{N-1} \\
&= h_N + x_{N-1} + x_N
\end{aligned} \tag{10.1}
$$

Dacă pentru (1) avem o soluție $x = (x_1, x_2, \ldots, x_N)$ care va produce înălțimea finală H, atunci adăugând / scăzând o valoare arbitrară d din toate valorile lui x vom obține o altă soluție $y = (x_1 + d, x_2 + d, \ldots, x_N + d)$ care va produce înălțimea finală $H + 2 \cdot d$.

Cazul N impar

Reducând fiecare dintre egalitățile din (1) obținem relațiile:

$$
\begin{aligned}
x_1 &= h_N - h_1 + x_{N-1} \\
x_2 &= h_1 - h_2 + x_N \\
x_3 &= h_2 - h_3 + x_1 \\
x_4 &= h_3 - h_4 + x_2 \\
&\ldots \\
x_{N-1} &= h_{N-2} - h_{N-1} + x_{N-3} \\
x_N &= h_{N-1} - h_N + x_{N-2}
\end{aligned} \tag{10.2}
$$

Fixăm $x_1 = 0$, din care rezultă x_3, apoi $x_5, \ldots, x_N$ (N impar), apoi $x_2, x_4, \ldots, x_{N-1}$ ($N - 1$ par). Nu putem accepta valori negative, astfel că vom calcula $x_{min} = \min\{x_k \mid 1 < k \leq N\}$ și vom scădea valoarea x_{min} din toate valorile x_k, $1 \leq k \leq N$. Această soluție va produce înălțimea minimă $h_{min} = h_1 + x_N + x_1$.

Cazul N par

Relațiile (2) determină două grupuri, ecuațiile cu necunoscute cu indici impari (3) și cele cu necunoscute cu indici pari (4):

$$
\begin{aligned}
x_1 &= h_N - h_1 + x_{N-1} \\
x_3 &= h_2 - h_3 + x_1 \\
x_{N-1} &= h_{N-2} - h_{N-1} + x_{N-3}
\end{aligned} \tag{10.3}
$$

$$
\begin{aligned}
x_2 &= h_1 - h_2 + x_N \\
x_4 &= h_3 - h_4 + x_2 \\
x_N &= h_{N-1} - h_N + x_{N-2}
\end{aligned} \tag{10.4}
$$

Se observă că, dacă adunăm toate relațiile din grupul (3), atunci toate variabilele x se reduc, rezultând egalitatea:

$$h_1 + h_3 + \cdots + h_{N-1} = h_2 + h_4 + \cdots + h_N \tag{10.5}$$

Observăm din nou că există soluție doar dacă datele de intrare respectă egalitatea (5). Dacă nu, atunci problema ne cere să eliminăm un număr minim de cărămizi. Așadar, răspunsul este dat de diferența în valoare absolută dintre cele două sume.

Rezolvăm separat fiecare grup de relații inițializând $x_1 = 0$ și respectiv $x_2 = 0$. Pentru (3) calculăm $x_{min1} = \min\{x_k \mid 1 \leq k \leq N, k \text{ impar}\}$ și apoi scădem x_{min1} din fiecare x_k, k impar. Pentru (4) calculăm $x_{min2} = \min\{x_k \mid 1 \leq k \leq N, k \text{ par}\}$ și apoi scădem x_{min2} din fiecare x_k, k par. Această soluție va produce înălțimea minimă $h_{min} = h_1 + x_N + x_1$.

Complexitate $O(N)$.

Subtaskul 5, soluția 2

Dacă problema are soluție, există și o soluție greedy. Parcurgem zidul de la 1 la N și, oricând avem $h_i > h_{i+1}$ creștem perechea (h_i, h_{i+1}) cu valoarea $h_i - h_{i+1}$. Astfel obținem un vector ordonat și reducem problema la subtaskul 2.

De aceea sunt suficiente două treceri prin vector. Complexitatea este $O(N)$.

10.9 Cod-sursă pentru problema Zid

```
#include <fstream>
#define NMAX 100002
using namespace std;
ifstream  fin("zid.in");
ofstream fout("zid.out");
int N, h[NMAX], i;
int solve3(){
    long long a[NMAX], i, j, c[NMAX], d, nc, s0, s1, hmin, cmin;
    s0=0; s1=0;
    for (i=1; i<=N; ++i){
        a[i]=h[i]; c[i]=0;
        if (i%2==0) s0=s0+h[i];
        else s1=s1+h[i];
    }
    if(N%2==0 && s0!=s1){
        fout<<abs(s0-s1)<<"\n";
        return 0;
    }
    for (i=2; i<=N-1; i=i+2)
        if(a[i-1]!=a[i]){
            d=a[i-1]-a[i];
            a[i]+=d;
            a[i+1]+=d;
            c[i]+=d;
        }
    for (i=1; i+2<=N-1; i=i+2){
        if (a[i]!=a[i+2]){
            d=a[i]-a[i+2];
            a[i+2]+=d;
            a[i+3]+=d;
            c[i+2]+=d;
        }
    }
    if (a[N-1]!=a[N]){
```

```
        d=a[N]-a[N-1];
        for (i=1; i<=N-2; i=i+2){
            a[i]+=d;
            a[i+1]+=d;
            c[i]+=d;
        }
    }
    if (N%2==1){
        cmin=1000000000;
        for (i=1; i<=N; ++i){
            if (c[i]<cmin) cmin=c[i];
            }
        for (i=1; i<=N; ++i) c[i]=c[i]-cmin;
    }
    else{
        cmin=1000000000;
        for (i=1; i<=N; i=i+2){
            if( c[i]<cmin) cmin=c[i];
            }
        for (i=1; i<=N; i=i+2) c[i]=c[i]-cmin;
        cmin=1000000000;
        for (i=2; i<=N; i=i+2){
            if (c[i]<cmin)
                cmin=c[i];
            }
        for (i=2; i<=N; i=i+2) c[i]=c[i]-cmin;
    }
    nc=0;
    for (i=1; i<=N; ++i){
        if (c[i]>0) nc++;
        a[i]=h[i];
    }
    for (i=1; i<=N; ++i){
        j=i+1;
        if (j>N) j=1;
        a[i]=a[i]+c[i];
        a[j]=a[j]+c[i];
    }
    hmin=a[1];
    fout<<hmin<<"\n";
    return 0;
}

int main(){
    fin>>N;
    for (i=1; i<=N; ++i) fin>>h[i];
    solve3();
    fout.close();
    return 0;
}
```

Capitolul 11

Barajul 2

11.1 Problema Lemmings

Propusă de: prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj

Lemmings este un joc video de strategie extrem de popular în anii 1990. Lemingii sunt niște rozătoare mici (șoricei), care viețuiesc mai ales în tundra din jurul Cercului Arctic. Sunt erbivori, hrănindu-se mai ales cu bulbi și rădăcini. Vizuinile lor au camere de odihnă, de hrană și camere de joacă.

Avem N camere numerotate de la 1 la N, dispuse circular. M camere conțin hrană, iar K lemingi sunt poziționați în camerele lor de odihnă care nu conțin hrană.

Dacă un leming se află în camera i și se deplasează spre dreapta, ajunge în camera $i + 1$ (cu circularitate: dacă este în camera N, merge în camera 1). Dacă se mișcă spre stânga, ajunge în camera $i - 1$ (cu circularitate: dacă este în camera 1, merge în camera N).

Trecerea dintr-o cameră în alta se face într-o unitate de timp.

Fiecare leming alege o direcție fixă de deplasare (stânga sau dreapta), și va merge constant în această direcție pentru următoarele T unități de timp. Lemingii nu staționează.

Dacă un leming se află într-o cameră cu hrană, el consumă instantaneu hrana respectivă și își continuă deplasarea.

Dacă doi lemingi se intersectează (se întâlnesc) aceștia dispar. Dacă se întâlnesc într-o cameră în care este hrană, unul dintre ei consumă hrana înainte de a dispărea.

Cerințe

Să se determine cantitatea maximă de hrană consumată de șoricei în T unități de timp.

Date de intrare

Fișierul de intrare `lemmings.in` conține pe prima linie numerele naturale N M K T, cu semnificația din enunț. Pe următoarea linie numerele celor M camere care conțin hrană, în ordine strict crescătoare. Ultima linie din fișier conține numerele celor K camere unde se află inițial lemingii, de asemenea în ordine strict crescătoare.

Date de ieșire

Fișierul de ieșire `lemmings.out` conține o singură linie pe care este scris un număr ce reprezintă cantitatea maximă de hrană consumată de lemingi.

Restricții

- $1 \leq N \leq 1\,000\,000$
- $1 \leq M, K \leq 20\,000$
- $1 \leq T \leq 500$

#	Puncte	Restricții
1	25	$1 \leq N \leq 100$, $1 \leq K \leq 20$
2	10	Distanța inițială între oricare doi lemingi este mai mare decât $2 \cdot T$
3	15	Nu există hrană și nici lemingi în camerele $1, 2, \ldots, T$
4	20	$1 \leq N \leq 100\,000$, $1 \leq M, K \leq 5\,000$
5	30	Fără alte restricții

Exemple

lemmings.in	lemmings.out
11 4 5 2 2 4 5 11 1 3 7 9 10	4
13 6 4 2 1 3 7 8 9 13 2 6 10 11	5

Explicații

În primul exemplu, avem $N = 11$ camere din care $M = 4$ conțin hrană (camerele 2 4 5 11). Cei $K = 5$ lemingi se află în camerele 1 3 7 9 10. După $T = 2$ unități de timp cantitatea maximă de hrană consumată este egală cu 4. O soluție posibilă este ca toți lemingii să meargă spre dreapta.

În al doilea exemplu, o posibilă alegere a direcțiilor este:

- Lemingul aflat în camera 2 merge spre stânga.
- Lemingul aflat în camera 6 merge spre dreapta.
- Lemingul aflat în camera 10 merge spre stânga.
- Lemingul aflat în camera 11 merge spre dreapta.

11.2 Rezolvarea problemei Lemmings

Toate soluțiile de mai jos necesită abilitatea de a calcula cantitatea de hrană disponibilă într-un interval de camere. Putem precalcula această informație într-un vector de lungime n (sau dublat la $2n$, pentru a simplifica tratarea circularității). Notăm 1 pe pozițiile unde există hrană și 0 în rest, apoi calculăm sume parțiale. Acum cantitatea de hrană din intervalul $[st, dr]$ este diferența între sumele parțiale la pozițiile dr și $st - 1$.

Această tehnică adaugă un factor de $O(n)$ la timp și la memorie. Pentru glorie virtuală, putem evita acest factor dacă calculăm informația în $O(m + k + t)$, interclasând vectorii de hrană și

lemingi. Nu putem calcula hrana din orice interval arbitrar, dar putem calcula o informație care ne interesează, și anume: pentru intervalul $[st, dr]$ dintre doi lemingi, câtă hrană este consumată în cele patru scenarii posibile?

1. Lemingul stâng pornește spre stânga, iar cel drept spre stânga.
2. Lemingul stâng pornește spre stânga, iar cel drept spre dreapta.
3. Lemingul stâng pornește spre dreapta, iar cel drept spre stânga.
4. Lemingul stâng pornește spre dreapta, iar cel drept spre dreapta.

Subtask 1

Când $k \leq 20$ putem evalua toate cele 2^k posibilități de orientare. Pentru fiecare posibilitate putem calcula hrana consumată în $O(k)$, iar o soluție în $O(2^k \cdot k)$ bine scrisă va trece toate testele.

Putem și să evaluăm posibilitățile în ordinea codului Gray, caz în care între două posibilități evaluate consecutiv trebuie să schimbăm direcția unui singur leming. Astfel obținem complexitatea $O(2^k)$, dar această complexitate nu este necesară pentru obținerea punctelor pe subtask.

Subtask 2

Dacă distanța între oricare doi lemingi (consecutivi) este mai mare de $2t$, atunci fiecare leming are „teritoriul" său în care doar el ajunge la hrană. Deci este suficient să calculăm, pentru fiecare leming, maximul dintre hrana disponibilă la stânga și la dreapta, iar răspunsul este suma acestor maxime.

Acest subtask poate ajuta la verificarea corectitudinii codului de sume pe interval, cu circularitate.

Subtask 3

Dat fiind că în intervalul $[1, t]$ nu există nici lemingi, nici hrană, putem considera vectorul liniar. Algoritmul corect este programarea dinamică descrisă mai jos, iar subtaskul poate ajuta la verificarea codului fără complexitatea suplimentară dată de circularitate.

Subtaskul 4

Soluțiile optime necesită reducerea problemei de la forma circulară la forma liniară. Pentru aceasta, este suficient să încercăm ambele orientări pentru unul dintre lemingi, să zicem primul. Apoi putem trata zona de $n - t$ camere rămase ca pe un vector normal (liniar). Calculăm rezultatul pentru acest vector și adăugăm hrana consumată de primul leming. Răspunsul este maximul dintre cele două variante.

Pentru forma liniară, o primă abordare definește $H[i][j][st/dr][st/dr]$ ca fiind cantitatea maximă de hrană pe care o pot mânca lemingii $i, i+1, \ldots, j$ în condițiile în care lemingul i pornește spre stânga / dreapta, iar lemingul j pornește spre stânga / dreapta. Practic, $H[i][j]$ este un cvadruplet.

Putem calcula recurent H fie încercând să împărțim $H[i]][j]$ în două porțiuni $H[i][r]$ și $H[r][j]$ pentru fiecare $r \in [i+1, j-1]$, fie extinzând intervalul $[i, j-1]$ cu o poziție la dreapta. De exemplu,

$$H[i][j][st][st] = \max \begin{cases} H[i][r][st][st] + H[r][j][st][st] \\ H[i][r][st][dr] + H[r][j][dr][st] \end{cases}$$

Observăm că lemingii i și j își păstrează orientările, iar pentru lemingul k încercăm ambele orientări. Cazul de bază este $j - i = 1$, care este un singur interval între doi lemingi consecutivi.

Acest algoritm duce la complexitatea $O(k^3)$ sau $O(k^2)$.

Subtaskul 5

Pentru punctaj maxim putem descrie o recurență mai simplă. Fie $x_1, x_2, \ldots x_k$ pozițiile lemingilor. Fie $S[x, y]$ cantitatea de hrană din intervalul închis $[x, y]$, cu convenția că, dacă $x > y$, atunci $S[x, y] = 0$. Fie $H[i][st/dr]$ cantitatea maximă de hrană consumată de primii i lemingi în condițiile în care lemingul i pornește spre stânga, respectiv spre dreapta. Atunci iau naștere patru cazuri similare, deoarece pentru fiecare orientare a lemingului i vom analiza cele două orientări posibile pentru lemingul $i - 1$.

$$H[i][st] = \max \begin{cases} H[i-1][st] + S[\max\{x_{i-1}, x_i - t\}, x_i] \\ H[i-1][dr] + S[\max\{x_{i-1} + t, x_i - t\}, x_i] \end{cases}$$

În cuvinte, lemingul i va consuma hrană spre stânga, nu mai mult de t poziții și fără a depăși lemingul $i - 1$ sau hrana consumată de acesta. Similar,

$$H[i][dr] = \max \begin{cases} H[i-1][st] + S[x_i, \min\{x_i + t, x_{i+1}\}] \\ H[i-1][dr] + S[x_i, \min\{x_i + t, x_{i+1}\}] \end{cases}$$

În cuvinte, lemingul i va consuma hrană spre dreapta, nu mai mult de t poziții și fără a depăși lemingul $i + 1$. Subliniem că nu ne preocupăm de situația în care lemingul $i + 1$ se îndreaptă spre stânga. Vom trata acest caz (lemingii i și $i + 1$ se îndreaptă unul spre celălalt) din perspectiva lemingului $i + 1$.

Motivul pentru care această recurență este corectă este că lemingul i nu poate fi influențat de decizia lemingilor $1, 2, \ldots, i - 2$. Lemingul $i - 1$ îi blochează pe toți aceștia, indiferent în ce orientare se află.

Acest algoritm are complexitatea $O(k)$, iar complexitatea totală este $O(k + m + t)$ sau $O(k + m + t + n)$ în funcție de implementarea sumelor pe interval.

11.3 Cod-sursă pentru problema Lemmings

```cpp
#include <bits/stdc++.h>
using namespace std;

ifstream  fcin("lemmings.in");
ofstream fcout("lemmings.out");

int n, m, k, t, Max, x;
int sp[2000003];
int s[20003];
int dp[20003][2];

int main()
{
    fcin >> n >> m >> k >> t;
    for (int i=1; i<=m; i++)
    {
```

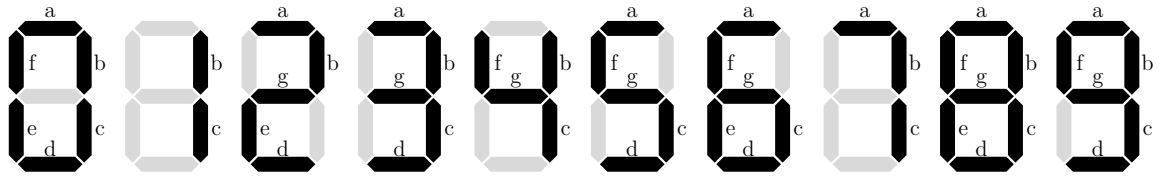
```
    fcin >> x;
    sp[x] = 1;
}
for (int i=1; i<=n; i++)
    sp[i] += sp[i-1];
///pentru susurinta dublez vectorul, sa nu mai fac i%n +1
for (int i=n+1;i<=2*n;i++)
    sp[i] = sp[i-1] + (sp[i-n] - sp[i-n-1]);

for (int i=1; i<=k; i++)
    fcin >> s[i];

///cazul 1: lemmingsul 1 spre stanga
int val = s[1] + n;
int poz = val - t;
poz = max(poz, s[k]+1);

dp[1][0] = sp[val] - sp[poz-1];
for (int i=2; i<=k; i++)
{
    int val1 = s[i] - t;
    int val2 = s[i-1] + t;
    int Max = max(val1, s[i-1]) - 1;
    if (val1 > val2)
        dp[i][0] = max(dp[i-1][0], dp[i-1][1]) + sp[s[i]] - sp[Max];
    else ///avem coliziune
    {
        int intersect = min(s[i-1] + t, s[i]);
        dp[i][0] = max(dp[i-1][0]+sp[s[i]]-sp[Max], dp[i-1][1]+sp[s[i]]-sp[intersect]);
    }
    if (i != k)
    {
        int Maxup = min(s[i]+t, s[i+1]);
        dp[i][1] = max(dp[i-1][1], dp[i-1][0]) + sp[Maxup] - sp[s[i]-1];
    }
}
dp[k][1] = max(dp[k-1][0], dp[k-1][1]);

int poz1 = s[1] + n - t;
poz1 = max(s[k]+1, poz1);
int poz2 = s[k] + t;
if (poz2 >= poz1)
{
    poz1--;
    dp[k][1] += sp[poz1] - sp[s[k]];
}
else
    dp[k][1] += sp[poz2] - sp[s[k]];

Max = max(dp[k][0], dp[k][1]);

///cazul 2: lemmingsul 1 spre dreapta
dp[1][0] = 0;
int Maxim = min(s[1] + t, s[2]);
dp[1][1] = sp[Maxim] - sp[s[1]-1];
for (int i=2; i<=k; i++)
{
    int val1 = s[i] - t;
    int val2 = s[i-1] + t;
    int Max = max(val1, s[i-1]) - 1;
    if (val1 > val2)
```

```
            dp[i][0] = max(dp[i-1][0], dp[i-1][1]) + sp[s[i]] - sp[Max];
        else ///avem coliziune
        {
            int intersect = min(s[i-1] + t, s[i]);
            dp[i][0] = max(dp[i-1][0]+sp[s[i]]-sp[Max], dp[i-1][1]+sp[s[i]]-sp[intersect]);
        }
        if (i != k)
        {
            int Maxup = min(s[i] + t, s[i+1]);
            dp[i][1] = max(dp[i-1][1], dp[i-1][0]) + sp[Maxup] - sp[s[i]-1];
        }
    }
    int pozitie = s[k] + t;
    pozitie = min(pozitie, s[1] + n);
    dp[k][1] = max(dp[k-1][0], dp[k-1][1]);
    dp[k][1] += sp[pozitie] - sp[s[k]];

    fcout << max({Max, dp[k][0], dp[k][1]});

    return 0;
}
```

11.4 Problema Mutare

Propusă de: prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil" Buzău

Un indicator numeric este un dispozitiv de afișaj electronic destinat afișării unei cifre zecimale. Acesta conține 7 segmente notate cu a, b, c, d, e, f, g. Afișarea unei cifre se face prin aprinderea segmentelor evidențiate din figura de mai jos, corespunzătoare fiecărei cifre:



Un număr natural N poate fi afișat utilizând unul sau mai multe indicatoare numerice.
Definim **mutarea** unui segment ca fiind succesiunea operațiilor de **stingere** a acestui segment și **aprinderea** sa în orice altă poziție, de pe oricare dintre indicatoarele numerice utilizate pentru afișarea numărului N.

Cerințe

Cunoscând un număr natural N, să se scrie un program care determină numerele care se pot afișa după mutarea unui **singur** segment, dintre segmentele utilizate pentru afișarea numărului N.

Date de intrare

Fișierul de intrare `mutare.in` conține pe prima linie numărul natural N.

Date de ieșire

Fișierul de ieșire `mutare.out` va conține pe prima linie numărul T ce reprezintă numărul total de numere care se pot obține prin mutarea unui singur segment, iar pe următoarele T linii, numerele obținute, în ordine **crescătoare**, câte un număr pe fiecare linie.

Restricții

- $0 \leq N < 10^{17}$
- doar numerele de o singură cifră pot începe cu cifra 0
- se garantează că pentru toate testele există cel puțin un număr care se poate obține prin mutarea unui singur segment.

#	Puncte	Restricții
1	16	$0 \leq N < 10$
2	28	$10 \leq N < 10^8$
3	56	$10^8 \leq N < 10^{17}$

Exemple

mutare.in	mutare.out
69	6 58 60 66 83 85 99

Explicații

$N = 69$
Se pot obține 6 numere prin mutarea unui singur segment și anume: 58, 60, 66, 83, 85 și 99.
De exemplu:

- numărul 60 se obține prin stingerea segmentului g al indicatorului cifrei 9 și aprinderea segmentului e, tot al indicatorului cifrei 9.
- numărul 83 se obține prin stingerea segmentului f al indicatorului cifrei 9 și aprinderea segmentului b al indicatorului cifrei 6

11.5 Rezolvarea problemei Mutare

Soluția 1 „brute-force”

Se construiește o structură de date care poate memora starea fiecărui segment al unui indicator și apoi se extinde această structură pentru mai multe indicatoare. Dacă un segment este aprins, în structură se va memora 1 și respectiv 0 dacă segmentul este stins. Se determină apoi numărul de cifre ale lui N, pe care să-l notăm cu nrcifre și se transformă fiecare cifră a numărului N într-un element al structurii. Se parcurg apoi toate numerele care au exact nrcifre și se păstrează doar acelea care diferă față de N printr-o singură mutare. Pentru a rezolva această problemă trebuie realizate funcții specifice pentru:

- comparare a două indicatoare
- comparare a două numere sub forma lor echivalentă din structură
- transformarea unui număr din forma sa structurată în număr de tip long long

Pentru a compara două numere sub forma lor echivalentă din structura se poate folosi operatorul XOR. Soluția poate obține aproximativ 40 puncte.

Soluția 2 „perechi de cifre”

O altă variantă, mai eficientă, se poate realiza prin parcurgerea tuturor perechilor $1 \leq i \leq j \leq nrcifre$ având în vedere că se face o singură mutare, mutare care stinge un segment pe poziția i și-l aprinde pe poziția j. Pozițiile i și j pot fi chiar egale, caz în care mutarea unui segment se face în cadrul aceluiași indicator. Sunt necesare și alte funcții suplimentare față de varianta anterioară cum ar fi o funcție care să testeze dacă un indicator la care s-a aprins un segment reprezintă sau nu, un număr. Ordinul de complexitate al soluției este $O(nrcifre^2)$. Soluția obține 100 puncte.

11.6 Cod-sursă pentru problema Mutare

```
#include <fstream>
```

```cpp
#include <algorithm>
#include <cassert>
#define cmax 25
using namespace std;

ifstream fin("mutare.in");
ofstream fout("mutare.out");

struct indicator
{
    int seg[7];
};

struct numar
{
    int nr_cifre;
    indicator T[cmax];
};

indicator cz[10]={1,1,1,1,1,1,0,  // 0
                  0,1,1,0,0,0,0,  // 1
                  1,1,0,1,1,0,1,  // 2
                  1,1,1,1,0,0,1,  // 3
                  0,1,1,0,0,1,1,  // 4
                  1,0,1,1,0,1,1,  // 5
                  1,0,1,1,1,1,1,  // 6
                  1,1,1,0,0,0,0,  // 7
                  1,1,1,1,1,1,1,  // 8
                  1,1,1,1,0,1,1}; // 9

long long sol[10000];
int s = 0;

int compara_ind(indicator A, indicator B)
{
 for (int i = 0; i<= 6; i++)
      if (A.seg[i] != B.seg[i]) return 0;
 return 1;
}

int e_numar(numar A)
{int i, k, ok = 0 ;
 for (i = A.nr_cifre ; i > 0; i--)
      for (k = 0; k <= 9 ;k++)
          if (compara_ind(A.T[i],cz[k])) {ok++;break;}
    if (ok == A.nr_cifre) return 1;
            else return 0;
}

long long afisare_numar(numar A)
{int i, k;
 long long S = 0;
 for (i = A.nr_cifre ; i > 0; i-- )
     for (k = 0; k <= 9 ;k++)
         if (compara_ind(A.T[i],cz[k])) S = S*10 + k;
 return S;
}

long long N;

int main()
```

```
{long long CN;
 int uc,k,i,j,k1,k2;
 numar W,CW;
 fin >> N;
 // determin numarul si configuratia indicatoarelor utilizate pentru N
 k = 0;
 CN = N;
 do
   {
    uc = CN%10;
    W.T[++k] = cz[uc];
    CN /= 10;
   }
 while (CN);
 W.nr_cifre = k;

 //analizez toate perechile de forma (1<=i<=j<=nr_cifre) intre care se poate face o mutare
 for (i = W.nr_cifre; i >=1 ; i--)
     for (j = W.nr_cifre; j >=1 ; j--)
         {
         // analizez toate segmentele indicatorului i
         for (k1 = 0; k1 <= 6;k1++)
             if (W.T[i].seg[k1] == 1)
                {
                 // analizez toate segmentelor indicatorului j
                 for (k2 = 0; k2 <= 6;k2++)
                     if (W.T[j].seg[k2] == 0)
                        {
                          CW  =  W;
                          // efectueaza mutarea in numarul CW
                          CW.T[i].seg[k1] = 0;
                          CW.T[j].seg[k2] = 1;
                          // afisez numarul (daca e numar!! si daca prima sa cifra nu este 0
                          if (e_numar(CW))
                              if (CW.nr_cifre == 1) sol[++s] = afisare_numar(CW);
                                  else
                                  if (compara_ind(CW.T[CW.nr_cifre],cz[0])==0)
                                      sol[++s] = afisare_numar(CW);
                        }
                }
           }
 sort(sol + 1,sol + s + 1);
 fout << s <<"\n";
 for (i = 1; i <= s; i++)
      fout << sol[i] <<"\n";
 fin.close(); fout.close();
 return 0;
}
```

11.7 Problema Wall-E

Propusă de: stud. Rareș-Andrei Cotoi, Facultatea de Matematică și Informatică, Universitatea Babeș-Bolyai Cluj-Napoca

Roboțelul explorator Wall-E se deplasează pe o hartă reprezentată ca o succesiune de celule, numerotate de la 1 la N. Fiecare celulă are asociat un anumit nivel de energie.

Se numește *secvență* o succesiune de celule de pe hartă numerotate consecutiv. *Energia* unei secvențe este egală cu suma nivelurilor de energie asociate celulelor din care este formată secvența.

Wall-E poate modifica nivelul de energie al unor celule respectând următoarele două condiții:

- nivelul de energie al unei celule modificate crește cu un număr natural cuprins între 1 și X;
- suma tuturor valorilor cu care au crescut nivelurile de energie ale celulelor modificate este egală cu S.

După modificarea nivelurilor de energie, Wall-E este interesat de secvențele critice de lungime L. O secvență de lungime L este considerată *critică* dacă energia secvenței este minimă, în raport cu toate secvențele de lungime L existente pe hartă.

Cerințe

Determinați energia maximă a unei secvențe critice de lungime L, după ce Wall-E modifică convenabil nivelul de energie al unor celule, respectând condițiile din enunț.

Date de intrare

Fișierul de intrare `walle.in` conține pe prima linie numărul natural N, reprezentând numărul de celule de pe hartă. Pe a doua linie se află numerele naturale X, S și L cu semnificația din enunț. Pe ultima linie se află N numere naturale, reprezentând nivelurile de energie ale celulelor de hartă în ordinea numerotării acestora. Valorile scrise pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `walle.out` conține o singură linie pe care este scris răspunsul la cerință.

Restricții

- $1 \leq L \leq N \leq 10^5$
- Nivelul de energie al oricărei celule este un număr natural nenul $\leq 10^3$.
- $1 \leq X \leq 10^3$
- $1 \leq S \leq 10^5$

#	Puncte	Restricții
1	13	$1 \leq N \leq 1\,000$
2	10	$N > 1000$, $L = 1$
3	6	$N > 1000$, $S = 1$
4	10	$N > 1$, $X = 1$
5	28	$1001 \leq N \leq 10\,000$
6	33	Fără restricții suplimentare.

Exemple

walle.in	walle.out
6 3 5 3 1 2 3 6 5 4	11
5 2 3 1 3 1 4 7 2	3

Explicații

Exemplul 1 $N = 6$, $X = 3$, $S = 5$, $L = 3$. Wall-e poate modifica nivelul de energie al celulelor astfel:

- pentru celula 1 nivelul de energie crește cu 2;
- pentru celula 2 nivelul de energie crește cu 1;
- pentru celula 3 nivelul de energie crește cu 2.

Nivelurile de energie ale celulelor de pe hartă devin: 3 3 5 6 5 4. Secvențele de lungime $L = 3$ au energia

- 3+3+5=11
- 3+5+6=14
- 5+6+5=16
- 6+5+4=15

Există o singură secvență critică de lungime 3 și aceasta are energia 11, aceasta fiind valoarea maximă posibilă.

Exemplul 2 $N = 5$, $X = 2$, $S = 3$, $L = 1$. Wall-E poate modifica nivelul de energie al celulelor astfel:

- pentru celula 2 nivelul de energie crește cu 2;
- pentru celula 5 nivelul de energie crește cu 1.

Nivelurile de energie ale celulelor de pe hartă devin: 3 3 4 7 3. Secvențele de lungime $L = 1$ sunt reprezentate de valoarea fiecărei celule, deci secvențele critice de lungime L au energia 3, aceasta fiind valoarea maximă posibilă.

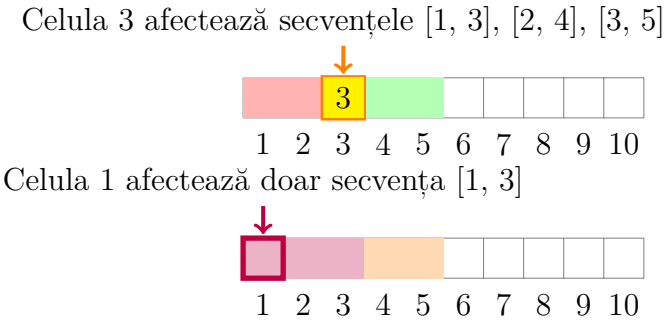
11.8 Rezolvarea problemei Wall-E

Cerința 1

O abordare posibilă este de a genera folosind metoda backtracking toate soluțiile obținute prin alocarea energiei în oricare dintre cele N celule, cu respectarea condiției date. O astfel de soluție obține aproximativ 20 de puncte, fiind o soluție viabilă doar în cazul unui N foarte mic.

Pentru a rezolva problema într-un mod eficient, vom folosi o abordare de tip Greedy. Încercăm să identificăm pentru o valoare potențială T, dacă este posibil să distribuim energia suplimentară astfel încât toate secvențele să aibă cel puțin energia T. Dacă putem atinge cel puțin suma T pentru fiecare secvență de lungime L, dar nu putem atinge cel puțin suma $T+1$, răspunsul final este T. Știind că T este suma elementelor dintr-o secvență de lungime L, remarcăm faptul că $T \in [0, sumaElem + S]$, unde $sumaElem$ este suma tuturor elementelor din celulele inițiale. Așadar, T poate fi determinat optim folosind căutare binară pe intervalul $[0, sumaElem + S]$.

Pentru fiecare sumă-candidat T, procesăm secvențele de lungime L de la stânga spre dreapta, iar pentru fiecare secvență care are nevoie de energie suplimentară (suma secvenței $< T$), adăugăm energie începând de la cea mai din dreapta celulă a secvenței, apoi continuăm spre stânga. Utilizăm această strategie deoarece o celulă care se află la poziția j afectează toate secvențele care o includ (de la cea care începe la poziția $j - L + 1$, până la cea care începe la poziția j). Astfel, prin adăugarea energiei în celula cea mai din dreapta a unei secvențe, maximizăm numărul de secvențe ulterioare care beneficiază de această creștere. De exemplu, dacă $N = 10$ și $L = 3$:



Pentru a calcula rapid sumele secvențelor inițiale, vom folosi un șir de sume parțiale sum, astfel:

$sum[1] \leftarrow \sum_{i=1}^{L} v[i]$
for $i \leftarrow 2$ to $n - L + 1$ **do**
    $j \leftarrow i + L - 1$
    $sum[i] \leftarrow sum[i-1] + v[j] - v[i-1]$

La momentul parcurgerii secvențelor de lungime L, putem folosi o coadă cu dublu acces (sau o stivă) pentru menținerea pozițiilor i în care mai putem adăuga energie ($v[i] < X$). Pentru o sumă-candidat T, un algoritm de verificare a validității lui T este:

function PUTEMATINGEENERGIA(T)
    Inițializăm o coadă cu dublu acces dq cu pozițiile 1 până la $L-1$
    for $i \leftarrow 1, L-1$ **do**
        $pus[i] \leftarrow 0$
    $R \leftarrow S$ ▷ *energia rămasă de distribuit*
    $P \leftarrow 0$ ▷ *energia suplimentară pe secvența curentă*
    for $st \leftarrow 1, n-L+1$ **do**
        $dr \leftarrow st+L-1$
        $dq.pushBack(dr)$ ▷ *adăugăm poziția dr în dq*
        $P \leftarrow P - pus[st-1]$ ▷ *elimin energia adăugată la poziția care iese din secvență*
        if $dq.front() = st-1$ **then**
            $dq.popFront()$ ▷ *eliminăm poziția care iese din secvență*
        if $sum[st]+P < T$ **then**
            $D \leftarrow T-P-sum[st]$ ▷ *deficitul pentru secvența curentă*
            while $dq.size() > 0$ **and** $D > 0$ **and** $R > 0$ **do**
                $w \leftarrow dq.back()$
                $A \leftarrow \min(D, \min(R, X-pus[w]))$ ▷ *cantitatea de energie de adăugat*
                $D \leftarrow D-A$
                $R \leftarrow R-A$
                $P \leftarrow P+A$
                $pus[w] \leftarrow pus[w]+A$
                if $pus[w] = X$ **then**
                    $dq.popBack()$ ▷ *eliminăm poziția dacă am atins limita X*
            if $D > 0$ **then**
                return $false$ ▷ *nu putem atinge energia T pentru această secvență*
    return $true$ ▷ *toate secvențele pot avea cel puțin energia T*

Complexitatea totală a algoritmului este $O((N+S) \cdot \log(suma\_initiala+S))$, unde căutarea binară are complexitatea $O(\log(suma\_initiala+S))$ și $O(N+S)$ (în cel mai rău caz) reprezintă timpul de verificare a validității unei sume-candidat.

11.9 Cod-sursă pentru problema Wall-E

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <deque>
#include <assert.h>

using namespace std;
ifstream f("walle.in");
ofstream g("walle.out");
typedef int64_t Int;
const int N=100010;
int n,m,k,L,stiva[N];
Int cnt,sum[N],pus[N],v[N],X,S;
bool ok(int64_t F)
{
    /// verific daca pot distribui cele S unitati astfel incat fiecare suma sa fie >=F
    deque<int> dq;/// deque cu pozitiile in care pot sa adaug unitati
    for(int i=1;i<L;i++)/// pot sa adug oriunde intre pozitiile 1 ... L-1
    {
```

```
        dq.push_back(i);
        pus[i]=0;
    }
    Int R=S;/// mai pot folosi
    Int P=0;/// ce suma suplimentara am pe intervalul [st,dr]
    for(int st=1,dr=L;dr<=n;st++,dr++)/// pe orice secventa de lungime L
    {
        dq.push_back(dr);pus[dr]=0;/// pentru sum[st] pot sa adaug si in pozitia dr
        P-=pus[st-1];
        if(dq.front()==st-1)/// nu mai pot sa adaug din pozitia
            dq.pop_front();
        if(sum[st]+P<F)
        {
            Int D=F-P-sum[st];/// sum[st] mai are nevoie de D unitati
            while(dq.size()>0 && D>0 && R>0) ///mai am de pus,mai am de unde, mai am unde
            {
                int w=dq.back();/// alegem ultima pozitie unde mai pot sa pun
                Int A=min(D,min(R,X-pus[w]));
                D-=A; /// mai am de pus cu A mai putin pentru a obtine F pentru sum[st]
                R-=A; /// in total mai pot pune cu A mai putin
                P+=A; /// pe intervalu [st,dr] am acum cu A mai mult
                pus[w]+=A; /// cele A unitati sunt puse la pozitia w
                if(pus[w]==X)/// daca acum am deja X la pozitia w elimin pozitia din coada
                    dq.pop_back();
            }
            /// am incercat dar nu am avut de unde
            if(D>0)
                return false;
        }
    }
    return true;
}
int main()
{
    f>>n>>X>>S>>L;
    for(int i=1;i<=n;i++)
        f>>v[i];
    for(int i=1;i<=L;i++)
        sum[1]+=v[i]; /// [1...L] sum[i] = suma(v[j] , j = i ... i+L-1
    Int hi=sum[1],lo=0LL;
    for(int i=2,j=L+1;j<=n;i++,j++)
    {

        sum[i]=sum[i-1]+v[j]-v[i-1];
        hi+=v[j];
    }
    hi+=S+1;
    while(hi-lo>1)
    {
        Int mi=(lo+hi)/2;
        if(ok(mi))
            lo=mi;
        else
            hi=mi;
    }
    g<<lo;
    return 0;
}
```

Partea a IV-a

Tabăra de pregătire a lotului național de informatică juniori

Zalău, 22-27 mai 2025

Capitolul 12

Barajul 3

12.1 Problema Allp

Propusă de: stud. Andrei Boacă, Facultatea de Informatică, Universitatea „Al. I. Cuza" Iași

Un șir de numere naturale $s_1, s_2, s_3, ..., s_k$ se numește palindrom dacă $s_i = s_{k-i+1}$ pentru orice i din intervalul $[1, k]$. Spunem că un șir de numere naturale $s_1, s_2, s_3, ..., s_k$ are proprietatea P dacă elementele sale pot fi reordonate astfel încât șirul rezultat să fie palindrom.

Un subșir al șirului $a_1, a_2, a_3, ..., a_n$ este un șir de forma $a_{p_1}, a_{p_2}, a_{p_3}, ..., a_{p_k}$ cu proprietatea că $1 \leq p_1 < p_2 < ... < p_k \leq n$.

Valoarea unui șir $s_1, s_2, s_3, ..., s_k$, notată cu $val(s_1, s_2, s_3, ..., s_k)$, este numărul de subșiruri nevide ale șirului s care au proprietatea P. Două subșiruri se consideră diferite dacă indicii selectați sunt diferiți, chiar dacă elementele sunt aceleași pe toate pozițiile.

Cerințe

Se dau un șir de numere naturale $v_1, v_2, v_3, ..., v_N$ și Q întrebări de forma (l, r). Pentru fiecare întrebare trebuie să aflați $val(v_l, v_{l+1}, v_{l+2}, ..., v_r)$. Deoarece această valoare poate fi foarte mare, se cere afișarea restului acesteia la împărțirea cu 998 244 353.

Date de intrare

Fișierul de intrare `allp.in` conține pe prima linie numerele N și Q. Următoarea linie conține N numere naturale nenule, elementele șirului v. Următoarele Q linii conțin fiecare câte două numere, l și r, conform descrierii din enunț.

Date de ieșire

Fișierul de ieșire `allp.out` conține Q linii, pe linia i aflându-se răspunsul la cea de a i-a întrebare din fișierul de intrare.

Restricții

- $1 \leq N, Q \leq 10^6$
- $1 \leq v_i \leq 10^9$ pentru orice i $(1 \leq i \leq N)$
- $1 \leq l \leq r \leq N$ pentru orice întrebare

#	Puncte	Restricții
1	7	$1 \leq N \leq 100$, $1 \leq Q \leq 5$, $r - l \leq 9$ pentru orice întrebare
2	10	$1 \leq v_i \leq 3$, $1 \leq N, Q \leq 400\,000$
3	21	$1 \leq Q \times N \leq 5\,000\,000$
4	12	$1 \leq v_i \leq 100$, $N \leq 400\,000$, $1 \leq Q \leq 2\,000$
5	17	$1 \leq N, Q \leq 60\,000$
6	15	$1 \leq N, Q \leq 120\,000$
7	18	Fără restricții suplimentare

Exemple

allp.in	allp.out
6 3 1 1 2 3 2 4 1 3 3 6 1 6	5 7 19

Explicații

Pentru prima întrebare subșirurile cu proprietatea P sunt formate din următoarele mulțimi de indici: $\{1\}$, $\{2\}$, $\{3\}$, $\{1, 2\}$, $\{1, 2, 3\}$.
Pentru a doua întrebare subșirurile cu proprietatea P sunt formate din următoarele mulțimi de indici: $\{3\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\{3, 5\}$, $\{3, 4, 5\}$, $\{3, 5, 6\}$.

12.2 Rezolvarea problemei Allp

Observație

Un șir are proprietatea P dacă există cel mult un număr cu frecvență impară. Acest lucru se datorează faptului că un palindrom de lungime pară nu poate avea un număr cu frecvență impară, iar un palindrom de lungime impară are un singur număr cu frecvență impară, cel din mijloc.

Soluția 1

Putem considera pe rând fiecare submulțime de indici din intervalul $[l, r]$ al unui query, numărând câte astfel de submulțimi respectă proprietatea P în forma rescrisă mai sus. Complexitatea acestei soluții este $O(Q \cdot (r-l) \cdot 2^{r-l+1})$, care este suficientă pentru primul subtask. Ea poate fi optimizată la $O(Q \cdot 2^{r-l+1})$ dacă facem trecerea între două submulțimi în $O(1)$.

Soluția 2

De acum vom considera numerele ca fiind „normalizate”, adică vom aduce toate numerele în intervalul $[1, N]$ și vom nota cu $VALMAX$ valoarea maximă din șir. Vom procesa query-urile pe rând și pentru fiecare vom determina frecvențele numerelor care apar în respectivul interval. Fie f_x frecvența numărului x. Numărul de moduri de a selecta un număr **par** de elemente cu valoarea x (dintre cele f_x disponibile) este egal cu

$$C_{f_x}^0 + C_{f_x}^2 + C_{f_x}^4 + ... = 2^{f_x - 1}$$

Similar, numărul de moduri de a selecta un număr **impar** de elemente cu valoarea x este

$$C_{f_x}^1 + C_{f_x}^3 + C_{f_x}^5 + ... = 2^{f_x - 1}$$

Aceste formule sunt consacrate, dar pot fi și deduse prin inducție sau chiar empiric, pe cazuri particulare de coeficienți binomiali ca (1, 4, 6, 4, 1) sau (1, 5, 10, 10, 5, 1).

Prin urmare, dacă selectăm un număr pentru a avea frecvență impară, sau dacă selectăm ca toate să aibă frecvențe pare, numărul de moduri de a obține o astfel de configurație este $\prod_x 2^{f_x - 1}$. Deci, răspunsul este $[\prod_x 2^{f_x - 1} \cdot (nrdif + 1)] - 1$, unde $nrdif$ este numărul de numere distincte (adică cu frecvența nenulă) din intervalul de query. Astfel, putem obține o complexitate $O(VALMAX * Q)$ care este suficientă pentru primele 4 subtask-uri.

Soluția 3

Observăm că $\prod_x 2^{f_x - 1} = 2^{r-l+1-nrdif}$. Deci, formula finală este $2^{r-l+1-nrdif} \cdot (nrdif + 1) - 1$. Deci, pentru fiecare interval trebuie să calculăm câte numere au frecvența mai mare decât 0. Acest lucru se poate face ușor folosind algoritmul lui Mo, obținându-se astfel o complexitate de $O((N + Q) \cdot \sqrt{N})$. Această soluție se încadrează în timp pe toate subtask-urile în afară de ultimul. Ea poate fi împinsă aproape de 100p cu optimizări ca: citirea rapidă; eliminarea operațiilor modulo; precalcularea puterilor lui 2.

Soluția 4

Pentru a optimiza soluția precedentă, vom grupa query-urile după capătul lor dreapta, iterând apoi prin pozițiile de la 1 la N și menținând într-un arbore indexat binar/arbore de intervale valoarea 1 pe pozițiile în în care se află ultima apariție a unui număr până la acel moment, respectiv 0 în rest. Astfel, numărul de numere distincte dintr-un interval se obține printr-o singură interogare de sumă pe interval în structura arborescentă. Acest articol detaliază algoritmul.

Complexitate finală: $O((N + Q) \cdot \log N)$.

12.3 Cod-sursă pentru problema Allp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ifstream fin("allp.in");
ofstream fout("allp.out");
const ll mod=998244353;
ll n, q, v[400005], nr;
map<ll,ll> nrm;
struct date { ll l, ind; };
vector<date> myq[400005];
ll sol[400005];
ll last[400005];
ll pw2[400005];
ll aib[400005];
```

```
ll lsb(ll x)
{
    return x&(-x);
}

void update(ll poz,ll val)
{
  for  (int i=poz;  i<=n;  i+=lsb(i)) aib[i]+=val;
}

ll suma(ll poz)
{ll rez=0;
 for  (int i=poz;  i>=1;  i-=lsb(i))   rez+=aib[i];
 return rez;
}

int main()
{ios_base::sync_with_stdio(false);  fin.tie(0);
 pw2[0]=1;
 for (int i=1; i<=4e5; i++) pw2[i]=(pw2[i-1]*2LL)%mod;
 fin>>n>>q;
 vector<ll> vals;
 for (int i=1; i<=n; i++)
     {
        fin>>v[i];
        vals.push_back(v[i]);
     }
 sort(vals.begin(),vals.end());
 vals.erase(unique(vals.begin(),vals.end()),vals.end());
 for (int i=0; i<vals.size(); i++)
    {
     nr++;
     nrm[vals[i]]=nr;
    }
 for (int i=1; i<=n; i++) v[i]=nrm[v[i]];
 for (int i=1; i<=q; i++)
     {ll l,r;
      fin>>l>>r;
      myq[r].push_back({l,i});
     }
 for (int i=1; i<=n; i++)
     {
      if (last[v[i]]!=0) update(last[v[i]],-1);
      last[v[i]]=i;
      update(i,+1);
      for (date p:myq[i])
         {ll l=p.l;
          ll ind=p.ind;
          ll cnt=suma(i)-suma(p.l-1);
          ll rez=((cnt+1)*pw2[i-l+1-cnt])%mod;
          rez=(rez-1+mod)%mod;
          sol[ind]=rez;
         }
     }
 for (int i=1; i<=q; i++) fout<<sol[i]<<'\n';
 return 0;
}
```

12.4 Problema Powtop

Propusă de: prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil" Buzău

Definim o **putere** ca fiind un număr natural P cu proprietatea că există alte două numere naturale $A > 1$ și $B > 1$ astfel încât $P = A^B$. Exemple de puteri : $8 = 2^3$; $625 = 5^4$; $7\,776 = 6^5$.

Asupra unui un șir de N numere naturale S_i, $1 \leq i \leq N$, se aplică următorul algoritm:

- Termenii șirului S_i, $1 \leq i \leq N$, se transformă într-un alt șir cu $N-1$ înmulțind fiecare doi termeni consecutivi.
- Se reia operația anterioară până când se obține un șir format dintr-un singur termen.

De exemplu: $S = [1, 2, 3, 4] \rightarrow [2, 6, 12] \rightarrow [12, 72] \rightarrow [864]$

Cerințe

Se consideră $\boldsymbol{T}$ șiruri notate cu X_i, $1 \leq i \leq T$, de câte N numere naturale fiecare. Pentru fiecare dintre cele T șiruri X_i, $1 \leq i \leq T$, se aplică algoritmul descris mai sus atât pentru șirul dat cât și pentru cele $N-1$ permutări circulare către **stânga** ale șirului X_i, $1 \leq i \leq T$.

Să se determine pentru fiecare șir X_i, $1 \leq i \leq T$, care dintre termenii obținuți sunt **puteri**.

Date de intrare

Fișierul de intrare `powtop.in` conține pe primul rând numerele naturale T și N, iar pe următoarele T linii câte N numere naturale ale șirului X_i, $1 \leq i \leq T$.

Date de ieșire

Fișierul de ieșire `powtop.out` trebuie să conțină T linii cu câte N numere de 0 sau 1 fiecare: 0 dacă termenul obținut prin aplicarea algoritmului nu este putere sau 1 dacă este putere. Numerele aflate pe aceeași linie trebuie separate prin câte un spațiu.

Restricții

- $1 \leq T \leq 100$
- $2 \leq N \leq 50$
- $1 \leq X_i \leq 10^7, 1 \leq i \leq N$

#	Puncte	Restricții
1	40	$T = 1$
2	24	$2 \leq T \leq 50$
3	36	$51 \leq T \leq 100$

Exemple

powtop.in	powtop.out
2 4 2 6 3 12 3 8 16 9	0 0 1 0 1 0 0 0

Explicație

$T = 2$, $N = 4$ și avem două șiruri: $X_1 = [2, 6, 3, 12]$ și $X_2 = [3, 8, 16, 9]$.

Prin aplicarea algoritmului pentru primul șir se obține numărul 139 968, care nu este putere. Pentru următoarele 3 permutări circulare la stânga se obțin numerele 559 872, 248 832 și 62 208. Dintre acestea, doar $248\,832 = 12^5$ este putere.

Prin aplicarea algoritmului pentru al doilea șir se obține numărul $56\,623\,104 = 384^3$, care este putere. Pentru următoarele 3 permutări circulare la stânga se obțin numerele 71 663 616, 2 519 424 și 1 990 656, care nu sunt puteri.

12.5 Rezolvarea problemei Powtop

Soluția 1

Se poate demonstra că având un șir S_i, $1 \leq i \leq N$ de numere naturale pe care îl transformăm succesiv de $N - 1$ ori într-un alt șir format din **suma** oricăror 2 termeni consecutivi ai șirului precedent, numărul obținut la final poate fi calculat cu ajutorul unei expresii combinatoriale fără a mai fi nevoie să aplicăm succesiv cele $N - 1$ transformări. De exemplu pentru șirul [10, 20, 30] se obține succesiunea:

$$[10, 20, 30] \to [30, 50] \to [80]$$

Așadar

$$\begin{aligned} 80 &= 30 + 50 \\ &= 10 + 20 \cdot 2 + 30 \\ &= 10 \cdot C_2^0 + 20 \cdot C_2^1 + 30 \cdot C_2^2 \end{aligned}$$

Prin inducție matematică se poate demonstra că numărul TOP din finalul aplicării algoritmului este egal cu:

$$TOP = S_1 \cdot C_{n-1}^0 + S_2 \cdot C_{n-1}^1 + \cdots + S_n \cdot C_{n-1}^{n-1} \tag{12.1}$$

Să analizăm ce se întâmplă când schimbăm operația de **adunare** cu aceea de **înmulțire**. Se poate observa că proprietatea de mai sus se transferă exponenților din descompunerea în factori primi a fiecărui număr S_i.

Să analizăm același șir $[10, 20, 30] = [2 \cdot 5, 2^2 \cdot 5, 2 \cdot 3 \cdot 5]$ dar aplicând operația de înmulțire. Se obțin succesiv șirurile:

$$[2 \cdot 5, 2^2 \cdot 5, 2 \cdot 3 \cdot 5] \to [2^3 \cdot 5^2, 2^3 \cdot 3 \cdot 5^2] \to [2^6 \cdot 3 \cdot 5^4]$$

Se poate observa că **exponenții** respectă proprietatea (12.1):

$$\begin{array}{rcccccc} 6 & = & 1 \cdot C_2^0 & + & 2 \cdot C_2^1 & + & 1 \cdot C_2^2 \\ 1 & = & 0 \cdot C_2^0 & + & 0 \cdot C_2^1 & + & 1 \cdot C_2^2 \\ 4 & = & 1 \cdot C_2^0 & + & 1 \cdot C_2^1 & + & 1 \cdot C_2^2 \end{array}$$

Așadar se poate calcula valoarea TOP din finalul aplicării algoritmului făcând o descompunere a numerelor S_i, $1 \leq i \leq N$ în factori primi și aplicând proprietatea (12.1). Contribuția (exponentul) unui factor prim la produsul final este dată de suma exponenților săi proveniți din toate elementele vectorului rotit. Produsul final este putere dacă cmmdc-ul exponenților este mai mare decît 1. De exemplu, $2^6 \cdot 3^4 = (2^3 \cdot 3^2)^2 = 72^2$ este putere, pe cînd $2^5 \cdot 3^4$ nu este putere.

Rezolvarea problemei presupune parcurgerea următoarelor etape:

1. **Precalcularea combinărilor**. Se poate face în mai multe moduri însă cel mai eficient este cu triunghiul lui Pascal.
2. **Precalcularea** exponenților din descompunerea în factori primi a fiecărui număr S_i, $1 \leq i \leq N$ în vederea folosirii lor repetate la fiecare permutare circulară.
3. **Determinarea exponenților** numărului TOP pentru fiecare permutare circulară a șirului dat utilizând expresia (12.1).
4. **Determinarea cmmdc** al exponenților anterior calculați pentru a determina dacă numărul TOP este putere.
5. **Permutarea circulară** către **stânga** a șirului S_i, $1 \leq i \leq N$.

Complexitatea soluției are trei componente:

- Factorizarea celor TN numere. Putem face acest lucru brut, în $O(TN\sqrt{VALMAX})$ sau cu ciurul lui Eratostene în $O(VALMAX \log VALMAX)$. Prima metodă este mai eficientă, mai ales dacă colectăm în prealabil numerele prime pînă la $\sqrt{VALMAX}$.
- Aflarea factorizării produsului final. Aceasta necesită $O(T \cdot N^2 \cdot \log N)$: pentru fiecare test, pentru fiecare rotație și pentru fiecare element, procesăm fiecare divizor al elementului.
- Testul de putere (aflarea cmmdc-ului exponenților). Acesta necesită $O(TN^3 \log N)$: pentru fiecare test, pentru fiecare rotație și pentru cei $O(N \log N)$ factori primi distincți facem o operație cmmdc. Știm că cmmdc-ul face un număr logaritmic de pași, iar valorile pe care operează (exponenții) provin din combinări de N, deci sînt exponențiali în N. De aceea, teoretic trebuie să socotim cmmdc-ul ca avînd cost $O(N)$.

În practică, dominantă este factorizarea, care merită implementată eficient.

Soluția 2

Iată o soluție diferită, care nu necesită găsirea formulei combinatorice. Calculăm produsele în mod naiv. Întrucît ele vor depăși rapid `long long`, factorizăm termenii șirului și îi reprezentăm ca pe liste de perechi (bază, exponent), ordonate după bază. Atunci ca să calculăm produsul a două numere, trebuie să interclasăm cele două liste (factorizări). Cînd întîlnim aceeași bază în ambele factorizări, adunăm exponenții. De exemplu:

$$(2^3 \cdot 7^2 \cdot 11^1) \times (3^1 \cdot 7^4 \cdot 13^2) = (2^3 \cdot 3^1 \cdot 7^6 \cdot 11^1 \cdot 13^2)$$

Astfel, calculăm o matrice A de produse de formă triunghiulară. Pe prima linie așezăm elementele șirului, iar $A_{l,c} = A_{l-1,c} \cdot A_{l-1,c+1}$. Atunci, în $A_{N,1}$ vom obține factorizarea produsului final. Ca și mai sus, calculăm cmmdc-ul exponenților din această factorizare ca să decidem dacă produsul este putere.

Această soluție are complexitatea $O(T \cdot N^4 \cdot \log N)$: pentru fiecare test și fiecare rotație, pentru fiecare din cele $O(N^2)$ produse calculate, interclasează două factorizări care, spre final, vor ajunge la $O(N \log N)$ factori. Soluția va obține circa 56 de puncte.

Putem reduce complexitatea dacă calculăm simultan răspunsul pentru **toate** rotațiile. Pe fiecare linie a matricei calculăm toate celulele, inclusiv ultima celulă, căreia îi dăm valoarea $A_{l,N} =$

$A_{l-1,N} \cdot A_{l-1,1}$. Atunci în cele N celule ale ultimei linii vom obține exact produsele celor N rotații.

Această soluție are complexitatea $O(T \cdot N^3 \cdot \log N)$, egală cu Soluția 1. Ea se comportă bine în practică, de exemplu pentru că este greu de construit un vector care chiar să conțină $O(N \log N)$ factori primi distincți.

12.6 Cod-sursă pentru problema Powtop

```
#include <bits/stdc++.h>
#define nmax 51
#define pmax 10000001
#define lmax 20
#define ll long long
using namespace std;
ifstream fin("powtop.in");
ofstream fout("powtop.out");
struct numar
{
    int x;
    int b[lmax];
    int e[lmax];
};

ll comb[nmax][nmax], p[pmax];
numar w[nmax];

ll cmmdc(ll a, ll b)
{
    if (b == 0) return a;
    else return cmmdc(b, a%b);
}

void combinari()
{
  comb[0][0] = 1; comb[1][0] = 1; comb[1][1] = 1;
  for (int i = 2; i <= 50; i++)
      {
       comb[i][0] = 1;
       for (int j = 1; j <= i; j++)
            comb[i][j] = comb[i-1][j] + comb[i-1][j-1];
      }
}

void desc(int n, numar w[nmax])
{ll t, fact, k;
 for (int i = 1; i <= n; i++)
     {t = w[i].x;
      // descompun t in factori primi
      k = 0; fact = 0;
          while (t%2==0) {t /= 2;fact++;}
          if (fact)
             { w[i].b[++k] = 2; w[i].e[k] = fact; }
        ll j = 3;
    while (j*j <= t)
          {
           if (t%j==0)
              {fact = 0;
               while (t%j==0)
                     {
                      t /= j;
```

```
                    fact++;
                   }
               if (fact)
                    {w[i].b[++k] = j; w[i].e[k] = fact; }
              }
           j += 2;
          }
     if (t > 1)
        { w[i].b[++k] = t; w[i].e[k] = 1; }
     w[i].b[0] = k; w[i].e[0] = k;
    }
}

void amplific(numar x, int n, int k)
{
 for (int i = 1; i <= x.e[0]; i++)
      p[x.b[i]] += x.e[i]*comb[n][k];
}

int main()
{int n, i, j, l, k, T;
 ll power;
 numar aux;
 // precalculez combinarile cu triunghiul lui Pascal
 combinari();
 // citire date de intrare
 fin >> T >> n;
 for (l = 1;l <= T; l++)
     {
      for (i = 1; i <= n; i++) fin >> w[i].x;
          desc(n,w);
      for (k = 1; k <= n; k++)
          {
           // initializez doar pozitiile unde sunt nr. prime
           for (i = 1; i <= n; i++)
               for (j = 1; j <= w[i].b[0]; j++)
                    p[w[i].b[j]] = 0;
           // amplific exponentii
           for (i = 1; i <= n; i++)
               amplific(w[i], n - 1, i - 1);
           // verific daca toti exponentii formeaza o putere
               power = 0;
           for (i = 1; i <= n; i++)
               for (j = 1; j <= w[i].b[0]; j++)
                   if (p[w[i].b[j]])
                       power = cmmdc(power,p[w[i].b[j]]);
           // afisez rezultatul
           if (power > 1)   fout << 1 << " ";
               else fout << 0 << " ";
           // permut circular la stanga
           aux = w[1];
          for (i = 2; i <= n; i++) w[i-1] = w[i];
          w[n] = aux;
          }
    fout <<"\n";
    }
    return 0;
}
```

12.7 Problema Sumgcd

Propusă de: prof. Mihai Bunget, Colegiul Național Tudor Vladimirescu Târgu-Jiu

Se dă un șir A cu N termeni numere naturale nenule. Pentru fiecare termen A_i, cu $i \geq 2$, definim $B_i = \max\{\gcd(A_i, A_1), \gcd(A_i, A_2), \cdots, \gcd(A_i, A_{i-1})\}$ și $S(A) = B_2 + B_3 + \cdots + B_N$, unde $\gcd(x, y)$ este cel mai mare divizor comun al numerelor x și y.

Cerințe

1. Să se determine $S(A)$.
2. Să se determine o permutare P a numerelor de la 1 la N pentru care $S(C)$ este maximă, unde C este șirul al cărui termeni sunt definiți prin $C_i = A_{P_i}$ pentru orice i de la 1 la N.

Date de intrare

Pe prima linie a fișierului `sumgcd.in` se află numerele T și N, unde T reprezintă numărul cerinței. Pe a doua linie se află N numere naturale nenule reprezentând termenii șirului A.

Date de ieșire

În fișierul `sumgcd.out` se va afișa $S(A)$ pentru $T = 1$, iar pentru $T = 2$ se vor afișa termenii permutării P, separați prin spații. Dacă există mai multe soluții, o puteți afișa pe oricare.

Restricții

- $1 \leq N \leq 100\,000$
- $1 \leq A_i \leq 1\,000\,000$
- La cerința 2, dacă $S(C)$ nu este maximă, se acordă punctaj parțial egal cu $\left(\frac{S(C)}{S(C')}\right)^4 \cdot 50\%$ din punctajul testului, unde C' este șirul pentru care S(C') e maximă.

#	Puncte	Restricții
1	8	$T = 1,\ 1 \leq N \leq 1\,000$
2	17	$T = 1,\ 1 \leq N \leq 100\,000$
3	18	$T = 2,\ 1 \leq N \leq 9$
4	57	$T = 2,\ 1 \leq N \leq 100\,000$

Exemple

sumgcd.in	sumgcd.out
1 5 12 7 15 21 20	16
2 5 12 7 15 21 20	1 5 3 4 2

Explicație

Pentru primul test avem $S(A) = \gcd(12, 7) + \gcd(15, 12) + \gcd(21, 7) + \gcd(20, 15) = 16$.
Pentru al doilea test avem permutarea (1, 5, 3, 4, 2) ce corespunde șirului $C = (12, 20, 15, 21, 7)$, iar $S(C) = \gcd(12, 20) + \gcd(20, 15) + \gcd(12, 21) + \gcd(7, 21) = 19$.

12.8 Rezolvarea problemei Sumgcd

Subtask 1

Pentru a calcula suma $S(A)$ se parcurge șirul A și pentru fiecare element A_i, cu $i \geq 2$, se calculează numărul B_i prin parcurgerea șirului A între indicii 1 și $i-1$ și calcularea celui mai mare divizor corespunzător.

Complexitate $O(N^2)$.

Subtask 2

Se determină divizorii numărului A_1 și se marchează cu 1 existența acestor divizori într-un vector de vizitare. Pentru fiecare termen din șirul A, începând cu al doilea, se determină divizorii acestuia și se reține cel mai mare divizor care a fost vizitat anterior. Acest divizor se adaugă la suma $S(A)$ și se actualizează vectorul de vizitare cu toți divizorii termenului curent.

Determinarea divizorilor lui A_i se poate face fie prin parcurgerea numerelor de la 1 la $\sqrt{A_i}$ și verificarea dacă acestea divid pe A_i, fie prin determinarea factorilor primi din descompunere și generarea divizorilor prin metoda backtracking.

Complexitate $O(N \cdot \sqrt{V})$ sau $O(V \cdot \log V)$ în funcție de metoda de determinare a divizorilor fiecărui termen al șirului A, unde V este valoarea maximă a valorilor termenilor din șirul A.

Subtask 2 - soluție alternativă

Observație: să presupunem că valoarea X apare în A pe pozițiile $i_1, i_2, \ldots, i_k$ (poziții în ordine crescătoare). Atunci $B_{i_2}, B_{i_3}, \ldots, B_{i_k}$ vor fi toate egale cu X. Cu alte cuvinte toate aparițiile valorii X în vector vor avea B_i asociat egal chiar cu X, mai puțin prima apariție.

De aceea vom calcula doar valorile din vectorul B ale primelor aparții ale valorilor din A. Pentru aceasta vom reține într-un vector caracteristic $P[]$ indicele primei aparții ale valorilor din A. Astfel $P[X]$ = indicele primei aparții ale valorii X.

În continuare vom folosi un algoritm tip ciur al lui Eratostene. Pentru fiecare valoare X din ciur vom parcurge valorile $D = X, 2X, 3X, \ldots$, etc. Pentru fiecare valoare D ne întrebăm dacă există printre valorile de la intrare consultând vectorul $P[]$. Vom reține două lucruri: numărul K de valori D care apar în A precum și M, indicele minim al aparției unei valori D.

Dacă K este cel puțin 2 înseamnă că vom avea valori la intrare al căror număr B este cel puțin X și marcăm acest lucru. Mai exact, pentru toate valorile D găsite în A mai puțin cea de la indicele M vom marca B asociat ca fiind X.

La final însumăm valorile B. Complexitate: $O(N + V \log V)$. Iată pseudocodul, considerând vectorul A citit:

Algoritmul 1: Calcul vector caracteristic $P[]$

```
Inițializează P[] la zero
for i ← 1 to N do
    if P[A[i]] = 0 then
        P[A[i]] ← i
    else
        B[i] ← A[i]
```

Algoritmul 2: Calcul vector $B[]$

```
Inițializează B[] la 1
V ← maximul valorilor din A
for X ← 2 to V do
    K ← 0                                   ▷ numărul de numere din A divizibile cu X
    M ← N + 1                               ▷ indice minim al unui număr divizibil cu X (inițial infinit)
    for D ← X to V cu pas X do
        if P[D] > 0 then
            K ← K + 1
            if P[D] < M then atunci
                M ← P[D]
    if K > 1 then                           ▷ reparcurgem valorile D
        for D ← X to V cu pas X do
            if P[D] > 0 și P[D] > M then
                B[P[D]] ← X
```

Algoritmul 3: Calcul suma B

```
S ← suma valorilor din B
```

Subtask 3

Cum N are o valoare mică se poate folosi metoda backtracking pentru generarea permutărilor șirului A (sau `next_permutation` din STL) și calculul pentru fiecare permutare a numărului $S(A)$.

Complexitate $O(N! \cdot \log V)$.

Subtask 4

Faptul că pentru fiecare termen A_i, cu $i \geq 2$, trebuie să găsim un termen anterior care maximizează cel mai mare divizor comun ne sugerează că fiecare termen „se leagă" de un termen anterior, ceea ce indică existența unei structuri arborescente.

Această intuiție este corectă. Putem reprezenta șirul ca pe un graf complet cu N noduri în care fiecărui termen îi corespunde un nod, iar costul pe muchia (A_u, A_v) este $-\text{cmmdc}(A_u, A_v)$. În acest graf, un arbore parțial de cost minim (APM) va consta dintr-o colecție de muchii care minimizează suma cu minus a cmmdc-urilor, așadar va maximiza suma cmmdc-urilor.

Astfel se poate aplica algoritmul lui Kruskal sau Prim pentru a determina arborele parțial de cost minim.

Implementare cu Algoritmul lui Kruskal

Se determină mai întâi divizorii numerelor din șirul A și pentru fiecare divizor se formează o listă cu indicii termenilor care au acest divizor.

Inițial fiecare termen al șirului A se consideră ca făcând parte dintr-o mulțime cu un element, acesta fiind considerat rădăcina unui arbore. Se parcurg descrescător valorile de la V la 1 (acestea fiind posibile costuri ale muchiilor) și, pentru fiecare valoare, dacă în lista corespunzătoare se află cel puțin două noduri se adaugă o muchie între primul nod din listă și oricare alt nod din listă numai în cazul în care cele două noduri fac parte din mulțimi (subarbori) diferite. De fiecare

dată se adaugă la suma $S(A)$ valoarea muchiei și se unesc cele două mulțimi (subarbori) prin subordonarea rădăcinii unui subarbore la rădăcina celuilalt.

În momentul în care au fost adăugate $N-1$ muchii și corespunzător costul muchiei la suma $S(A)$ atunci se obține valoarea maximă a acestei sume.

Complexitate $O(V \cdot \log V)$.

Implementare cu algoritmul lui Prim

Algoritmul lui Prim este un Greedy pe care îl putem intui natural și dacă nu găsim reducerea formală la APM. Inițializăm permutarea cu orice element, apoi de $N-1$ ori adăugăm la permutare acel element care maximizează cmmdc-ul între elementele deja alese (cele din permutare) și cele încă nealese. Aceasta este fix ideea algoritmului lui Prim, care pentru grafuri clasice se implementează cu heap-uri. Numim elementele alese „jumătatea stângă", iar numerele încă nealese „jumătatea dreaptă".

Pentru problema de față menținem, pentru jumătățile stângă și dreaptă, două structuri de date similare. Pentru fiecare divizor d, ținem minte un contor: câte numere din mulțime se divid cu d. Atunci, prin definiție, cmmdc-ul maxim între cele două jumătăți este d-ul maxim care are un contor nenul în ambele jumătăți. Apar, deci, trei nevoi:

1. Să găsim rapid d-ul maxim.
2. Să găsim un număr x din jumătatea dreaptă care se divide cu d.
3. Să-l mutăm pe x în stânga.

Pentru (2), este suficient să ținem, pentru fiecare divizor d, lista elementelor din dreapta care sunt multipli de d. Memoria este $O(N \log N)$, acceptabilă.

Pentru (3), trebuie să iterăm prin divizorii lui x (de exemplu recursiv), să decrementăm contoarele din dreapta și să le incrementăm pe cele din stânga.

Pentru (1), să observăm că în stânga contoarele doar cresc, iar în dreapta doar scad. Atunci putem menține un AIB sau un arbore de intervale care stochează 1 pe pozițiile d unde contoarele sunt pozitive în ambele jumătăți. Când mutăm primul multiplu de d din dreapta în stânga, scriem 1 în AIB pe poziția d, iar când mutăm ultimul multiplu de d, scriem 0 în AIB pe poziția d. Cu această informație, răspunsul la (1) este dat de poziția maximă a unui 1 în AIB, pe care o putem găsi cu o căutare binară în $O(\log MAX\_VAL)$.

12.9 Cod-sursă pentru problema Sumgcd

```
#include<bits/stdc++.h>
#define N 1000003
using namespace std;
ifstream f("sumgcd.in");
ofstream g("sumgcd.out");
int t, n, m, x, y, i, j, Vmax, no;
int p[N], a[N], viz[N], bif[N], gr[N], ta[N], vizit[N], pr[1201], b[N];
vector<int> vec[N], arb[N], diviz[N];

void divizori()
{
    int i, j, w, h, r, q, u;
    no = 0;
    for (i = 2; i <= 1200; i++)
        if (p[i]==0)
```

```
        {
            no++;
            pr[no] = i;
            j = i+i;
            while(j <= 1200)
            {
                p[j] = 1;
                j = j+i;
            }
        }

    for(i = 1; i <= n; i++)
        if(vizit[a[i]]==0)
        {
            w = a[i];
            vizit[a[i]] = 1;
            diviz[w].push_back(1);
            vec[1].push_back(i);
            for(j = 1; pr[j]*pr[j] <= w; j++)
                if(w%pr[j]==0)
                {
                    h = pr[j];
                    r = 1;
                    q = diviz[a[i]].size();
                    while(w%h==0)
                    {
                        r = r*h;
                        for(u = 0; u < q; u++)
                        {
                            diviz[a[i]].push_back(diviz[a[i]][u]*r);
                            vec[diviz[a[i]][u]*r].push_back(i);
                        }
                        w = w/h;
                    }
                }
            if(w > 1)
            {
                q = diviz[a[i]].size();
                for(u = 0; u < q; u++)
                {
                    diviz[a[i]].push_back(diviz[a[i]][u]*w);
                    vec[diviz[a[i]][u]*w].push_back(i);
                }
            }
        }
        else for(u = 0; u < diviz[a[i]].size(); u++)
                vec[diviz[a[i]][u]].push_back(i);
}

void solve1()
{
    int h,i,j,m;
    long long s;
    for(i = 0; i < diviz[a[1]].size(); i++)
        viz[diviz[a[1]][i]] = 1;
    s = 0;
    for(i = 2; i <= n; i++)
    {
        m = 0;
        for(j = 0; j < diviz[a[i]].size(); j++)
        {
```

```
            h = diviz[a[i]][j];
            if((viz[h]==1)and(h > m)) m = h;
            viz[h] = 1;
        }
        s += m;
    }
    g << s << "\n";
}

void dfs(int nod)
{
    g << nod << " ";
    bif[nod] = 1;
    for (int h = 0; h < arb[nod].size(); h++)
        if (bif[arb[nod][h]]==0) dfs(arb[nod][h]);
}

void solve2()
{
    int h,u,nr,nrgr,e,w,z,c,s;
    long long suma;

    for (h = 1; h <= n; h++)
    {
        gr[h] = h;
        b[h] = 1;
    }
    nr = 0; suma = 0;
    for (h = 1000000; h >= 1; h--)
        if ((vec[h].size() > 1)and(nr < n-1))
        {
            x = vec[h][0];
            u = x;
            while (u != 0)
            {
                e = u;
                u = ta[u];
            }
            for (c = 1; c < vec[h].size(); c++)
            {
                y = vec[h][c];
                w = y;
                while (w != 0)
                {
                    s = w;
                    w = ta[w];
                }
                if (s != e)
                {
                    if(b[s]<b[e]) b[e] += b[s];
                    else
                    {
                        b[s] += b[e];
                        swap(s,e);
                    }
                    nr++;
                    suma += h;
                    arb[x].push_back(y);
                    arb[y].push_back(x);
                    ta[s] = e;
                    u = x;
```

```
                    while (u != e)
                    {
                        z = ta[u];
                        ta[u] = e;
                        u = z;
                    }
                    u = y;
                    while (u != e)
                    {
                        z = ta[u];
                        ta[u] = e;
                        u = z;
                    }
                }
            }
        }
    dfs(1);
}


int main()
{
    f >> t >> n;
    for (i = 1; i <= n; i++)
    {
        f >> a[i];
        if (a[i] > Vmax) Vmax = a[i];
    }
    divizori();
    if (t==1) solve1();
    else solve2();
    return 0;
}
```

Capitolul 13

Barajul 4

13.1 Problema Căsuța

Propusă de: stud. Victor Botnaru, Facultatea de Automatică și Calculatoare, Universitatea Politehnica București

Gigel vrea să își construiască o căsuță în București. El vrea să aleagă o amplasare și o formă bună pentru fundația casei.

Pentru simplitate, considerăm Bucureștiul ca fiind un pătrat cu latura de N metri, situat într-un sistem de coordonate cartezian, având colțul stânga-jos în punctul (0,0), și colțul dreapta-sus în punctul (N,N).

Gigel, excentric de fel, vrea ca forma fundației căsuței să fie una aparte, anume, trebuie să respecte următoarele condiții:

- Forma este un triunghi dreptunghic, având catetele paralele cu axele sistemului și colțul drept poziționat în dreapta-jos.
- Vârfurile formei au coordonatele întregi.
- Lungimea catetei orizontale este un multiplu al lungimii catetei verticale.

În București, unele zone sunt mai valoroase decât celelalte. Mai simplu, Gigel cunoaște pentru fiecare metru pătrat din București dacă acesta este valoros sau nu, prin intermediul unei matrice binare V, de dimensiuni $N \times N$. Elementul $V_{i,j}$ are valoarea 1 sau 0 după cum aria acoperită de pătratul cu colțul stânga-jos $(j-1,\ i-1)$ și colțul dreapta-sus (j, i) este sau nu valoroasă. Numerotarea liniilor, respectiv a coloanelor începe de la 1.

Definim valoarea unei forme ca fiind suma ariilor intersecțiilor dintre formă și fiecare pătrat valoros din București.

Cerințe

Se dau T variante de fundație care respectă condițiile impuse de Gigel. Pentru fiecare, afișați valoarea formei acesteia.

Date de intrare

Pe prima linie a fișierului `casuta.in` se află două numere, N și T, cu semnificația din enunț. Urmează N linii a câte N cifre de 1 sau de 0, **fără spații între ele**, valorile matricei V, indexate

de la 1. Pe următoarele T linii se află câte patru întregi, X_1, Y_1, X_2, Y_2, reprezentând coordonatele vârfurilor ipotenuzei unei variante de fundație.

Date de ieșire

În fișierul `casuta.out` se va afișa câte un număr real pe fiecare dintre cele T linii, anume valoarea acelei forme. Numerele trebuie afișate cu 5 zecimale exacte.

Restricții

- $1 \leq N \leq 1\,000$
- $1 \leq T \leq 1\,000\,000$
- $0 \leq X_1 < X_2 \leq N, 0 \leq Y_1 < Y_2 \leq N$, pentru toate formele din intrare.
- Fie $\mathcal{L}$ lungimea celei mai mari catete verticale, și fie $\mathcal{A}$ suma ariilor tuturor formelor din intrare.

#	Puncte	Restricții
1	10	$V[i][j] = 1,\ 1 \leq i, j \leq N$
2	10	$1 \leq \mathcal{A} \leq 5.000.000$
3	10	$1 \leq T \leq 5.000$
4	10	$\mathcal{L} = 1,\ 1 \leq N \leq 300$
5	10	$1 \leq N \leq 300$
6	20	$\mathcal{L} = 1$
7	30	Fără restricții suplimentare.

Exemple

casuta.in	casuta.out
5 3 10000 01010 00000 00000 00100 0 0 2 2 1 0 5 2 2 4 5 5	1.00000 0.25000 0.16666

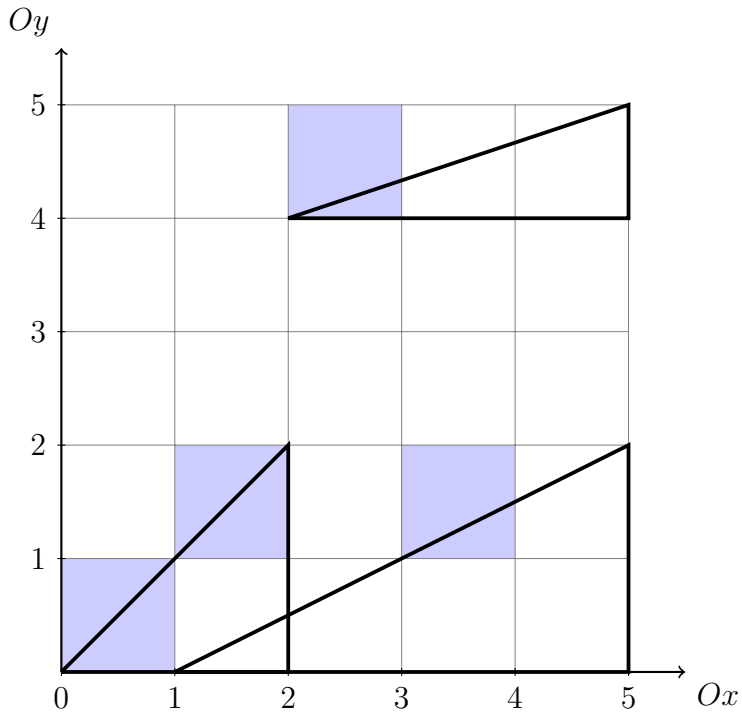
Explicație

Exemplul corespunde diagramei de mai jos. Cu mov am notat pătratele de arie valoroasă.

13.2 Rezolvarea problemei Căsuța

Subtaskul 1

Dacă toată matricea este valoroasă, atunci întreaga arie a fiecărui triunghi este valoroasă, deci trebuie doar să tipărim, pentru fiecare triunghi, valoarea



$$\frac{(x_2 - x_1)(y_2 - y_1)}{2}$$

Subtaskul 2

Dacă aria tuturor triunghiurilor este mică, ne permitem să calculăm, pentru fiecare celulă din submatricea acoperită de triunghi, aria acelei celule, care fie va fi complet acoperită, fie va avea forma unui trapez.

Subtaskul 3

Următoarele subtaskuri necesită precalcularea sumelor parțiale în V, pe linii, pe coloane sau pe două dimensiuni, după caz. Acestea ne oferă în $O(1)$ valoarea unui dreptunghi de mărime $k \times 1$, $1 \times k$, respectiv $k_1 \times k_2$.

De asemenea, în toate subtaskurile următoare vom defini **panta** unui triunghi ca fiind $(x_2 - x_1)/(y_2 - y_1)$, invers decât în sensul strict geometric.

Dacă T este suficient de mic, ne permitem o abordare în $O(TN)$, mai exact $O(\text{suma\_lungimilor})$. Pentru un triunghi dat, baleiem x de la x_1 la x_2 și adăugăm la răspuns aria valoroasă pe coloana $[x-1, x]$. Această coloană constă dintr-o coloană de pătrate (pentru care avem precalculată suma valoroasă) și dintr-un trapez inclus într-o celulă a matricei.

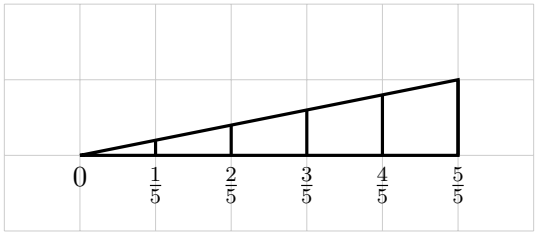
Această implementare poate trece și alte teste, în funcție de eficiență. Iată, de exemplu, o rutină care nu efectuează decât două împărțiri per triunghi, una pentru a afla panta și una la final, pentru a calcula aria.

```
double query(int x1, int y1, int x2, int y2) {
  const int slope = (x2 - x1) / (y2 - y1);
  int r_area = 0; // aria dreptunghiurilor de sub triunghiuri
  int t_area = 0; // aria trapezelor înmulțită cu numitorul 2*slope
```

```
  for (int x = x1, y = y1; x < x2; x += slope, y++) { // un triunghi
    for (int frac = 0; frac < slope; frac++) {
      // Trapez cu colțul din stînga-jos în (x+frac,y) de lățime 1
      // și cu bazele de înălțime frac/slope și (frac+1) / slope
      int right_x = x + frac + 1;
      r_area += col[y][right_x] - col[y1][right_x];
      // Valoarea celulei care conține trapezul,
      // ca diferență de sume parțiale.
      int rich = col[y + 1][right_x] - col[y][right_x];
      t_area += (frac * 2 + 1) * rich;
    }
  }

  return r_area + t_area / (2.0 * slope);
}
```

Subtaskurile 4 și 6

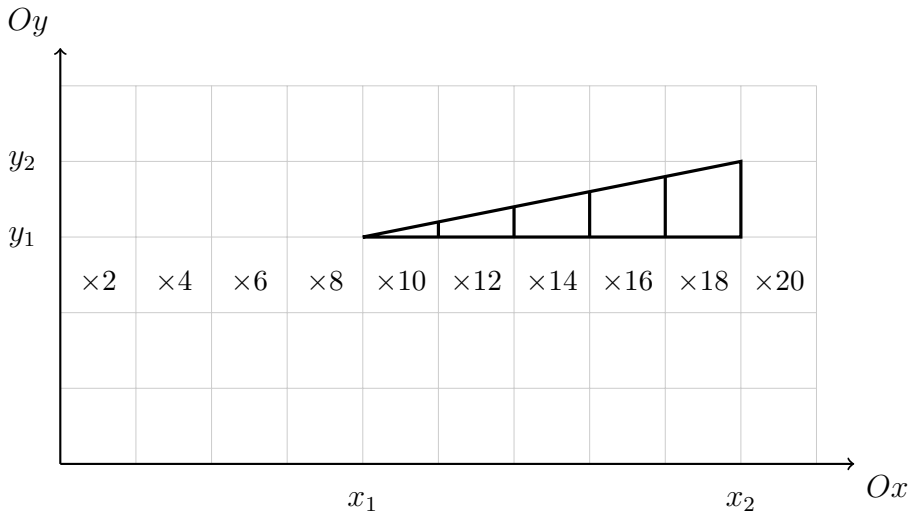
Să vedem cum putem calcula în $O(1)$ valoarea unui triunghi de înălțime 1. Vom exemplifica pe un caz particular ($x_2 - x_1 = 5$) pentru a nu încărca formulele.



Cele 5 trapeze au ariile 1/10, 3/10, 5/10, 7/10 și 9/10. Valoarea triunghiului este însă dată de suma produselor dintre aceste arii și celulele corespunzătoare din matrice. Pentru concizie, fie $c_1, c_2, \ldots, c_n$ valorile matricei V pe linia y_2 (linia cu triunghiul). Atunci dorim să calculăm:

$$\frac{1}{2 \cdot 5} \cdot (1 \cdot c_{x_1+1} + 3 \cdot c_{x_1+2} + 5 \cdot c_{x_1+3} + 7 \cdot c_{x_1+4} + 9 \cdot c_{x_2})$$

Acum, să spunem că triunghiul este așezat la $x_1 = 4$.



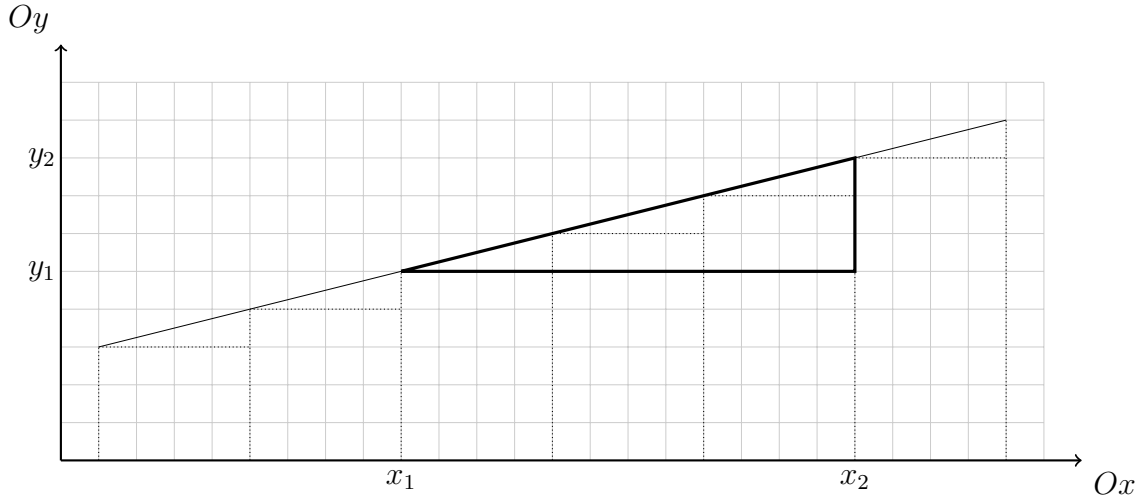
Să precalculăm pe linia curentă, pe fiecare coloană x, suma

$$S_x = 2 \cdot c_1 + 4 \cdot c_2 + \cdots + 2 \cdot x \cdot c_x$$

Atunci $S_{x_2} - S_{x_1}$ este o sumă pe intervalul acoperit de triunghi. Această sumă este, pe toate pozițiile, mai mare cu 9 ($= 2(x_2 - x_1) - 1$) decît aria valoroasă dorită. De aceea, putem scădea de 9 ori aria valoroasă a dreptunghiului $(x_1, y_1) - (x_2, y_2)$.

Subtaskurile 4 și 5

Acum putem aborda și triunghiurile de înălțime mai mare decât 1. Când N este suficient de mic, ne permitem o soluție în $O(T + N^3)$. Să considerăm triunghiurile în ordinea pantei (le putem sorta în timp liniar deoarece există cel mult N pante distincte). Pentru o pantă fixată p și pentru fiecare punct (x, y) să calculăm valoarea totală a triunghiurilor de $p \times 1$ care încap în matrice la stânga lui x, precum și a dreptunghiurilor de sub ele (așadar, valoarea unui trapez maximal).



Putem calcula această matrice în $O(N^2)$. Valoarea trapezului care se termină la (x, y) provine din:

- valoarea triunghiului $(x - p, y - 1) - (x, y)$;
- valoarea dreptunghiului $(x - p, 0) - (x, y - 1)$;
- valoarea trapezului anterior, care se termină la $(x - p, y - 1)$.

Atunci putem calcula valoarea unui triunghi cu panta p în $O(1)$ ca fiind:

- valoarea trapezului care se termină la (x_2, y_2);
- minus valoarea trapezului care se termină la (x_1, y_1);
- minus valoarea dreptunghiului $(x_1, 0) - (x_2, y_1)$

Subtaskul 7

Când N este suficient de mare, tratăm triunghiurile diferit după cum panta lor este mai mare sau mai mică decât o pantă $P = O(\sqrt{N})$. În practică orice valoare a lui P între 25 și 64 poate obține punctaj maxim.

Când panta este mică, procedăm ca la subtaskul anterior și obținem complexitatea $O(T+N^2\sqrt{N})$ pentru toate triunghiurile care au pante mici.

Când panta este mare, remarcăm că fiecare triunghi se compune din cel mult N/P triunghiuri de înălțime 1, deci putem însuma, folosind teoria de la subtaskul 6, toate aceste triunghiuri și dreptunghiurile de sub ele, în $O(\sqrt{N})$ per interogare.

Rezultă o complexitate totală de $O((T + N^2)\sqrt{N})$.

13.3 Cod-sursă pentru problema Căsuța

```
#include <iostream>
#include <stdio.h>
#include <iomanip>
#include <algorithm>
#pragma gcc optimize "Ofast"
using namespace std;

const int maxs = 32;
const int maxn = 1005;
const int maxq = 1e6+5;
const long long precizion = 1e5;

int scara_sum[maxn][maxn];
int prefix_sum[maxn][maxn];
int trapez_sum[maxn][maxn];
long long ans[maxq];
int x_one[maxq],  y_one[maxq],  x_two[maxq],  y_two[maxq];
int panta[maxq], srt[maxq];
int n, t;
bool v[maxn][maxn];

static bool cmp(const int a,const int b)
{
    return panta[a]<panta[b];
}

int get_rectangle_sum(int st, int dr, int h)
{
    return prefix_sum[h][dr]-prefix_sum[h][st];
}

int get_triangle_sum(int st, int dr, int h)
{
    int ans=scara_sum[h][dr]-scara_sum[h][st];
    int drept=        prefix_sum[h][dr]-prefix_sum[h][st]-
                prefix_sum[h-1][dr]+prefix_sum[h-1][st];
    ans-= (2*st) * drept;
    return ans;
}

//construieste precalcularea de panta p
void do_trapez_sum(int panta)
{
 for (int i=1; i<=n; i++)
      for (int j=1; j<=n; j++)
          trapez_sum[i][j]=0;
 for (int i=1; i<=n; i++)
     for (int j=panta; j<=n; j++)
        {
          int tri= get_triangle_sum(j-panta,j, i);
          int drept = get_rectangle_sum(j-panta, j, i-1);
          drept=drept*2*panta;
          trapez_sum[i][j]=trapez_sum[i-1][j-panta] + tri + drept;
        }
}
```

```
int main()
{
 #ifndef LOCAL
    freopen("casuta.in", "r", stdin);
    freopen("casuta.out", "w", stdout);
 #endif
 cin.tie(0); cout.tie(0);
 ios::sync_with_stdio(0);
 cin>>n>>t;

 string s;
 for (int i=1; i<=n; i++)
     {
      cin>>s;
      for (int j=1; j<=n; j++)
          {
            v[i][j]=(s[j-1]=='1');
            //sume partiale in scara
            scara_sum[i][j]=scara_sum[i][j-1]+(2*j-1)*v[i][j];
            prefix_sum[i][j]=         prefix_sum[i-1][j]+prefix_sum[i][j-1]-
                                 prefix_sum[i-1][j-1]+v[i][j];
          }
    }

 for (int i=1; i<=t; i++)
     {
      cin>>x_one[i]>>y_one[i]>>x_two[i]>>y_two[i];
      panta[i]= (x_two[i]-x_one[i])/(y_two[i]-y_one[i]);
      srt[i]=i;
    }

//ca sa nu precalculam pentru toate pantele initial,
//sortam dupa panta, si precalculam doar atunci cand e nevoie
 sort(srt+1,srt+t+1,cmp);

 //le facem pe cele cu panta mai mica decat sqrt
 int i=1;
 while (i<=t&&panta[srt[i]]<=maxs)
       {
        if (i==1|| panta[srt[i-1]] != panta[srt[i]])
            do_trapez_sum(panta[srt[i]]);
        int j=srt[i];
        ans[j]=trapez_sum[y_two[j]][x_two[j]] - trapez_sum[y_one[j]][x_one[j]];
        ans[j]=ans[j]-(2*panta[j])*get_rectangle_sum(x_one[j],x_two[j],y_one[j]);
        ans[j]*=precizion;
        ans[j]/=(2*panta[j]);
        i++;
    }
 //le facem pe cele cu panta mai mare decat sqrt
 for (; i<=t; i++)
     {
      int j=srt[i];
      int my_x = x_one[j];
      int my_y = y_one[j];
      ans[j] = 0;
      while (my_y < y_two[j])
            {
            int my_tri = get_triangle_sum(my_x, my_x + panta[j], my_y+1);
            int my_square = get_rectangle_sum(my_x, my_x+panta[j], my_y);
            my_square -= get_rectangle_sum(my_x, my_x+panta[j], y_one[j]);
            ans[j] += my_square * 2 * panta[j] + my_tri;
```

```
            my_y++;
            my_x+=panta[j];
        }
      ans[j] *= precizion;
      ans[j]/=(2*panta[j]);
    }
 for (int i=1; i<=t; i++)
      printf("%lld.%05lld\n",ans[i]/precizion, ans[i]%precizion);
 return 0;
}
```

13.4 Problema Nrk

Propusă de: stud. Alin-Gabriel Răileanu, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza" Iași
prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian, Drobeta Turnu Severin

Se dau 4 numere naturale N, K, A, B.

Cerințe

Să se determine cel mai mare număr format din N cifre, obținut prin alipirea a K numere distincte din intervalul $[A, B]$.

Date de intrare

Fișierul de intrare `nrk.in` conține pe singura linie cele 4 numere N, K, A, B, separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `nrk.out` conține o singură linie pe care este scris numărul cerut.

Restricții

- $1 \leq K \leq N \leq 200$
- $1 \leq A \leq B < 10^{200}$
- Se garantează că pentru datele de test există soluție.

#	Puncte	Restricții
1	5	$B - A \leq 10, 1 \leq A \leq B \leq 500$
2	6	$K = 2$, $1 \leq A \leq B \leq 1\,000$
3	11	A și B au același număr de cifre.
4	14	$K = 3$, $1 \leq A \leq B \leq 10^9$
5	64	Fără restricții suplimentare.

Exemple

nrk.in	nrk.out	Explicații
4 3 8 29	9829	Se alipesc numerele 9, 8, 29, în această ordine.
11 4 4 2397	99999998997	Se alipesc numerele 99, 999, 998, 997.
12 3 500 10000	999999989997	Se alipesc numerele 9999, 9998, 9997.

13.5 Rezolvarea problemei Nrk

Soluția oficială - Programare dinamică

Notă: Pentru obținerea răspunsului corect într-o complexitate timp decentă, este necesară colectarea numerelor care pot „candida" la un loc în răspuns, iar apoi sortarea acestora.

Sortarea reprezintă elementul-cheie în cadrul acestei soluții, întrucât în funcție de criteriul folosit, se pot obține rezultate diferite.

Criteriul corect de ordonare a două șiruri x și y este: $x \leq y$ dacă și numai dacă $x \oplus y \leq y \oplus x$, unde $x \oplus y$ denotă concatenarea numerelor x și y în ordinea aceasta.

Intuiție: Când rescriem criteriul d.p.d.v. matematic, obținem:

$$\begin{gathered} x \oplus y \leq y \oplus x \iff \\ x \cdot 10^{|y|} + y \leq y \cdot 10^{|x|} + x \iff \\ \frac{x}{10^{|x|} - 1} \leq \frac{y}{10^{|y|} - 1} \end{gathered}$$

criteriu în care fiecare număr este independent. Prin $|x|$ ne referim la lungimea numărului x.

Evident, putem sorta folosind acest criteriu toate numerele naturale din intervalul $[A, B]$, dar asta ar duce la TLE din pricina volumului mare.

Observație: Din fiecare grupă de dimensiune a numerelor ($10^L \leq x < 10^{L+1}$) ne interesează doar cele mai mari $\frac{N}{L}$ numere (în cazul în care există). Dacă selectăm mai multe, depășim N cifre în total, deci nu vom folosi numerele selectate în plus.

Colectăm aceste numere într-un vector V, pe care îl sortăm **descrescător**. Acum avem o ordonare favorabilă a numerelor, ceea ce înseamnă că orice soluție va fi un subșir de lungime K al lui V cu suma lungimilor numerelor egală cu N.

În continuare vom utiliza următoarea structură de programare dinamică:

$dp[s][i][j]$ = cel mai mare număr de lungime i care se poate obține prin alipirea a j numere distincte dintre primele s

Pentru a adăuga fiecare număr $V[s] = NR$ în structură vom utiliza următoarea recurență:

```
for s ← 1 to lungime(V) do
    dp[s] ← dp[s − 1]                    ▷ atribuire de matrice, în caz că nu folosim NR
    for i ← N to |NR| do
        for j ← k to 1 do
            dp[s][i][j] ← max{dp[s][i][j], dp[s − 1][i − |NR|][j − 1] ⊕ NR}
```

În practică, ne dăm seama că nu ne este necesară o matrice tridimensională dp, ci ne este suficient un singur strat în care adunăm, pe rând, fiecare $V[s]$. În final, complexitatea algoritmului descris mai sus se calculează astfel:

- Partea de selectare: În total vom selecta cel mult $\sum_{i=1}^{i \leq N} \frac{N}{i} = O(N \cdot \log N)$ numere. Suma lungimilor numerelor va fi $O(N^2)$, căci la fiecare lungime ne oprim când depășim N cifre colectate. Generarea numerelor presupune decrementări în $O(1)$. Astfel complexitatea este $O(N^2)$.
- Partea de sortare: $O(N^2 \cdot \log N \cdot \log(N \cdot \log N)) = O(N^2 \cdot \log^2 N)$. Aceasta deoarece sortăm un vector cu $N' = N \log N$ numere făcând $O(N' \log N')$ comparații, iar fiecare comparație durează $O(N)$.
- Programarea dinamică: $O(K \cdot N^3 \cdot \log N)$. Aceasta deoarece completăm o matrice de dimensiuni $N \log N \times N \times K$, iar completarea fiecărei celule necesită o comparare de șiruri.

Complexitatea timp finală va fi $O(K \cdot N^3 \cdot \log N)$, mult amortizată din pricina lungimilor numerelor.

Complexitatea memorie va fi $O(N^2(K + \log N))$: un vector de $O(N \log N)$ șiruri de lungime $O(N)$ și matricea *dp* care reține $N \cdot K$ șiruri de lungime $O(N)$.

Optimizare: Pentru a reduce complexitatea timp, se pot menține coduri hash pentru prefixele numerelor, astfel compararea se poate face prin căutare binară pe rezultat în $O(\log |NR|)$, fapt care duce la o complexitate timp de $O(K \cdot N^2 \cdot \log^2 N)$.

13.6 Cod-sursă pentru problema Nrk

```
#include <fstream>
#include <string>
#include <vector>
#include <algorithm>
const int NMAX=205;

using namespace std;
ifstream cin("nrk.in");
ofstream cout("nrk.out");

string dp[NMAX][NMAX];

bool cmp(string a, string b)
{
    return (a+b)>(b+a);
}

string solve_dp(vector <string> v, int n, int k)
{
    sort(v.begin(), v.end(), cmp); ///sortare folosind criteriul
    int cnt=0, i, j;
    for(i=0; i<=n; i++) ///initializare dp
    {
        for(j=0; j<=k; j++)
        {
            dp[i][j]=" ";
        }
    }
    dp[0][0]=""; ///caz de baza
    for(auto s:v)
    {
        cnt++;
        for(i=n; i>=(int)s.size(); i--) ///setare lungime
        {
            for(j=min(k, cnt); j>=1; j--) ///setare #numere
            {
                ///asigurare sa existe state-ul anterior
                if(dp[i-s.size()][j-1]!=" ")
                {
                    ///recurenta
                    dp[i][j]=max(dp[i][j], dp[i-s.size()][j-1]+s);
                }
            }
        }
    }
    return dp[n][k];
}

void dif(string& s) ///decrementare
```

```
{
    int poz=s.size()-1;
    while(s[poz]=='0') s[poz--]='9';
    s[poz]--;
}

///formarea multimii de numere necesare
vector <string> form(int n, int k, string a, string b)
{
    vector <string> ans;
    int sz, cnt;
    for(; b.size()>a.size();)
    {
        sz=b.size();
        cnt=min(k, n/sz)+1;
        while(cnt-- && (int)b.size()==sz)
        {
            ans.push_back(b);
            dif(b);
        }
        if((int)b.size()==sz) ///daca nu am sarit deja in grupa cealalta
        {
            ///transform totul in 9 si merg in grupa de mai jos
            for(auto &i:b) i='9';
            b.pop_back();
        }
    }
    ///caz particular pentru ultima grupa
    sz=b.size();
    cnt=min(k, n/sz)+1;
    while(cnt--)
    {
        ans.push_back(b);
        if(a!=b) dif(b);
        else break;
    }
    return ans;
}

int main()
{
    int n, k;
    string a, b;
    vector <string> v;
    cin>>n>>k>>a>>b;
    v=form(n, k, a, b);
    cout<<solve_dp(v, n, k)<<'\n';
    return 0;
}
```

13.7 Problema Passepartout

Propusă de: instr. Cristian Frâncu, Nerdvana București

Passepartout face înconjurul lumii, care este reprezentată simplificat ca o banală matrice A cu $N \times N$ poziții. Fiecare poziție conține numărul țării din care face parte, între 1 și M, unde M este numărul de țări. Țările sunt contigue: el poate ajunge din orice punct al unei țări în orice alt punct al acelei țări, deplasându-se pe orizontală sau pe verticală doar prin poziții ale acelei țări. Unele poziții din matrice nu aparțin niciunei țări: ele conțin numărul 0.

Passepartout pleacă din colțul de sus-stânga, **ce nu aparține niciunei țări**, și se deplasează pe orizontală sau pe verticală cu scopul de a vizita toate țările, **în ordinea crescătoare a numărului de țară**. El poate trece oricând prin orice țară (inclusiv prin poziții cu numărul 0), dar consideră țara ca vizitată doar dacă a vizitat toate țările cu număr mai mic. Cu alte cuvinte *Passepartout* trebuie să viziteze, pe rând, o poziție a țării 1, apoi o poziție a țării 2, și așa mai departe până la o poziție a țării M.

Cerințe

Să se calculeze lungimea minimă a unui drum al lui *Passepartout*.

Date de intrare

Prima linie a fișierului de intrare `passepartout.in` conține N și M, respectiv numărul de linii și de coloane ale matricei și numărul de țări. Următoarele N linii conțin câte N numere ce semnifică țara din care face parte acea poziție. O poziție ce nu aparține niciunei țări este codificată cu 0.

Date de ieșire

Fișierul de ieșire `passepartout.out` va conține lungimea drumului minim pe care îl va parcurge *Passepartout* pentru a vizita toate țările în ordinea numărului de țară.

Restricții

- $5 \leq N \leq 1\,000$
- $1 \leq M \leq \min(150, N \times N - 1)$
- $0 \leq A_{ij} \leq M$
- În interiorul unei țări se poate ajunge din orice punct în orice alt punct.
- Se garantează că există M țări pe hartă (fiecare număr de la 1 la M apare cel puțin o dată în matrice).

#	Puncte	Restricții
1	3	$M = 1$
2	4	$M = 2$
3	5	$M = 3$
4	6	$N \leq 13$
5	11	$N \leq 230$
6	21	$N \leq 680$
7	29	$N \leq 850$
8	21	Fără restricții suplimentare.

Exemple

passepartout.in	passepartout.out	Explicații
5 4 0 1 1 1 1 2 1 1 0 3 2 1 1 3 3 2 3 3 3 0 4 4 3 3 3	8	Drumul lui Passepartout este cel marcat cu verde și roșu. Remarcați că el trece de două ori prin poziția marcată cu roșu (a doua și a patra poziție vizitată).
5 4 0 3 3 3 2 4 3 3 2 2 4 4 3 2 2 1 0 3 3 2 1 1 1 2 2	10	Drumul lui Passepartout este cel marcat cu verde.
8 9 0 6 6 6 6 4 4 4 1 6 7 8 8 8 4 4 1 7 7 9 9 4 4 4 1 1 7 7 9 4 4 5 1 7 7 9 9 9 5 5 1 7 2 2 9 5 5 5 1 2 2 3 3 5 5 5 1 1 2 2 3 3 5 5	28	Drumul lui Passepartout este cel marcat cu verde.

13.8 Rezolvarea problemei Passepartout

Definim distanța Manhattan între două poziții din matrice, (L_1, C_1) și (L_2, C_2) ca fiind $|L_2 - L_1| + |C_2 - C_1|$, unde $|A|$ este valoarea absolută a lui A.

Subtaskul 1

Când avem o singură țară răspunsul este distanța Manhattan de la (1, 1) la cea mai apropiată valoare 1, la care se adaugă 1 pentru poziția de pornire. Complexitate $O(N^2)$ timp și memorie.

Subtaskul 2

Când avem două țări, putem calcula distanța până la țara 1 conform primului subtask. Apoi, pentru fiecare valoare 1, calculăm distanța la cel mai apropiat 2. Pentru aceasta putem folosi un

algoritm BFS în matrice (zis și Lee) cu multiple puncte de pornire în toate valorile 2. Pe măsură ce atingem valori 1 marcăm distanțele.

În final răspunsul este minimul sumei celor două distanțe pentru toate valorile 1. Complexitate $O(N^2)$ timp și memorie.

Subtaskul 4

Când N este foarte mic putem folosi o parcurgere în adâncime, similară cu un backtracking, dar care nu eșuează: parcurgem, pe rând fiecare poziție 1. Pentru fiecare poziție 1, încercăm fiecare poziție 2 și așa mai departe. Complexitate $O((N^2/M)^M)$ ca timp, $O(N^2)$ memorie.

Subtaskul 5

O soluție de complexitate $O(N^5)$ sau $O(N^4 \times M)$ ar trebui să ia punctele acestui subtask. Iată o soluție $O(N^4 \times M)$: Pentru fiecare țară $2 \leq K \leq M$ și pentru fiecare pereche de coordonate (L_1, C_1) și (L_2, C_2), dacă (L_1, C_1) aparține țării $K-1$, iar (L_2, C_2) aparține țării K, atunci încercăm să îmbunătățim distanța până la (L_2, C_2) prin (L_1, C_1).

Subtaskul 6

Să presupunem că am calculat lungimile minime ale drumurilor până la toate pozițiile țării $K-1$. Dorim să calculăm minimele pentru pozițiile țării K. Pentru fiecare poziție (L, C) unde avem o valoare K vom calcula:

$$D[L][C] = \min\{D[L_i][C_i] + |L - L_i| + |C - C_i|\}$$

unde (L_i, C_i) sunt pozițiile în matrice ale țării $K-1$.

Pentru o implementare de complexitate $O(N^4/M)$ va trebui să colectăm pentru fiecare țară pozițiile unde apare valoarea ei în matricea inițială.

Această soluție va lua punctaj parțial. Pentru a lua toate punctele acestui subtask observăm că nu este necesar să luăm în considerare toate pozițiile unei țări, ci doar cele de pe frontieră.

O altă îmbunătățire este ca, la fiecare trecere de la țara $K-1$ la țara K, să luăm în considerare doar punctele din țara $K-1$ ce au valori minime ale drumului, relativ la vecinii lor. Aceste „puncte speciale" sunt suficiente deoarece orice drum de la o altă poziție din țara $K-1$ la orice poziție din țara K poate fi ajustat ca să treacă printr-un punct special. Această optimizare va trece și teste din subtaskul următor.

Subtaskul 7

Putem gândi problema puțin diferit: am putea aplica BFS pe matrice (zis și Lee). Dar cum procedăm când, în parcurgere, ajungem la țara 1? Am putea să calculăm distanțele la țara 1, apoi să reluăm un Lee modificat, în care pornim cu lungimile drumurilor sortate crescător. Când introducem o nouă valoare în coadă, o vom insera în ordine crescătoare. Vom folosi o coadă de priorități (heap). Algoritmul seamănă cu cel al lui Dijkstra, de drum minim în graf.

Pentru a nu complica algoritmul Lee, putem face altceva: atunci când dăm de o valoare 1, să ne deplasăm **în jos**. Ne putem imagina o matrice 3D, în care prima „felie" orizontală, cea de sus, corespunde țării 1, următoarea țării 2 și așa mai departe. Când parcurgem BFS această matrice,

în „felia” K, vom genera vecinii în același plan dacă poziția curentă nu are valoarea K în matricea originală. Altfel vom rămâne la aceeași poziție și vom trece la următoarea „felie”, $K + 1$.

Algoritmul trebuie implementat cu grijă pentru a nu depăși memoria. Complexitatea este $O(N^2 \cdot M)$ ca timp și memorie.

Subtaskul 8

Ne propunem să calculăm, pe rând, drumurile minime către țara 1, apoi către toate țările $2, 3, \ldots, M$. Să presupunem că am calculat drumurile minime până la țara $K - 1$ și dorim să le calculăm pentru țara K. Pentru orice astfel de poziție, drumul optim poate veni de la o poziție anterioară ce poate fi în direcția unuia din colțurile matricei. Putem, deci, evalua cele patru drumuri optime posibile, apoi selectăm minimul dintre ele. Vom arăta cum rezolvăm problema pentru colțul (1, 1).

Parcurgem matricea pe linii. Pentru fiecare poziție vom calcula drumul minim până la acea poziție, indiferent dacă acea poziție aparține țării K ce se dorește a fi calculată. Atunci:

$$P_1[L][C] = \begin{cases} D[L][C] & \text{dacă } A[L][C] = K - 1, \text{sau} \\ \min\{P_1[L-1][C] + 1, P_1[L][C-1] + 1\} & \text{altfel} \end{cases}$$

Similar vom calcula P_2, P_3 și P_4, lungimile minime pentru cele patru direcții. Parcurgerile se modifică pentru a calcula minimele parțiale corect.

La final calculăm matricea D pentru pozițiile unde avem valori K:

$$D[L][C] = \min\{P_1[L][C], P_2[L][C], P_3[L][C], P_4[L][C]\}$$

Complexitatea acestei soluții este tot $O(N^2 \times M)$ ca timp, dar $O(N^2)$ memorie. Constanta este mai mică, drept care va lua 100p.

Există și o soluție $O(N^2 \log N)$. Ideea este similară cu soluția anterioară, dar ne propunem să facem trecerea de la țara $K-1$ la țara K în $O(N \log N)$. Considerând că avem drumurile calculate pentru țara $K - 1$, dorim să calculăm drumurile pentru țara K. Vom proceda similar, împărțind problema în patru subprobleme, drumurile optime ce vin din cele patru direcții.

Din nou, să considerăm direcția colțului (1, 1).

Să considerăm următoarea structură vectorială $V[]$: să presupunem că pe coloana C avem poziții cu valori $K - 1$. Fie ele L_i, cu drumurile minime precalculate $D[L_i][C]$. Atunci, la poziția $V[C]$ vom memora $\min(D[L_i][C] - L_i - C)$.

Acum, parcurgem matricea pe linii. Atunci când la poziția (L, C) întâlnim valoarea $K - 1$ în A, vom actualiza elementul $V[C]$ cu minimul corespunzător. Atunci când la poziția (L, C) întâlnim valoarea K în A, răspunsul pentru $P_1[L][C]$ va fi:

$$P_1[L][C] = \min\{L + C + V[C_i]\} = L + C + \min\{V[C_i]\}$$

unde $C_i \leq C$. De ce? Deoarece drumul optim este:

$$P_1[L][C] = \min\{D[L_i][C_i] + L - L_i + C - C_i\} = L + C + \min\{D[L_i][C_i] - L_i - C_i\}$$

pentru $L_i \leq L$ și $C_i \leq C$. Deoarece introducem punctele $K-1$ în V prin parcurgere pe linii, ne asigurăm că ambele condiții sunt îndeplinite.

Dacă vom căuta minimul liniar, obținem un algoritm $O(N^3)$. Însă putem folosi o structură de calcul rapid al minimului pe intervale, cum ar fi un arbore de intervale sau un arbore indexat binar. Complexitatea scade la $O(N^2 \log N)$.

13.9 Cod-sursă pentru problema Passepartout

```
#include <bits/stdc++.h>
using namespace std;
ifstream fin("passepartout.in");
ofstream fout("passepartout.out");
const int INF=1e9;
struct point
{
        int x,y,val,d;
};
vector<point> v[155];
int n,m;
bool comp(point a,point b)
{
        if(a.x!=b.x)
                return a.x<b.x;
        return a.val<b.val;
}
int aib[2][1005];
void reset()
{
        for(int i=1;i<=n;i++)
                aib[0][i]=aib[1][i]=INF;
}
int lsb(int x)
{
        return x&(-x);
}
void update(int ind,int poz,int val)
{
        for(int i=poz;i<=n;i+=lsb(i))
                aib[ind][i]=min(aib[ind][i],val);
}
int query(int ind,int poz)
{
        int rez=INF;
        for(int i=poz;i>=1;i-=lsb(i))
                rez=min(rez,aib[ind][i]);
        return rez;
}
int main()
{        ios_base::sync_with_stdio(false); fin.tie(0);
        fin>>n>>m;
        for(int i=1;i<=n;i++)
                for(int j=1;j<=n;j++)
                {
                        int x;
                        fin>>x;
                        if(x!=0)
                                v[x].push_back({i,j,x,INF});
                }
```

```
	for(point &p:v[1])
		p.d=p.x-1+p.y-1;
	for(int z=2;z<=m;z++)
	{
		vector<point> pts;
		for(point p:v[z-1])
			pts.push_back(p);
		for(point p:v[z])
			pts.push_back(p);
		sort(pts.begin(),pts.end(),comp);
		reset();
		for(int i=0;i<pts.size();i++)
		{
			if(pts[i].val==z-1)
			{
				int val=pts[i].d-pts[i].x-pts[i].y;
				update(0,pts[i].y,val);
				val=pts[i].d-pts[i].x+pts[i].y;
				update(1,n-pts[i].y+1,val);
			}
			else
			{
				int val=query(0,pts[i].y)+pts[i].x+pts[i].y;
				pts[i].d=min(pts[i].d,val);
				val=query(1,n-pts[i].y+1)+pts[i].x-pts[i].y;
				pts[i].d=min(pts[i].d,val);
			}
		}
		reset();
		reverse(pts.begin(),pts.end());
		for(int i=0;i<pts.size();i++)
		{
			if(pts[i].val==z-1)
			{
				int val=pts[i].d+pts[i].x-pts[i].y;
				update(0,pts[i].y,val);
				val=pts[i].d+pts[i].x+pts[i].y;
				update(1,n-pts[i].y+1,val);
			}
			else
			{
				int val=query(0,pts[i].y)-pts[i].x+pts[i].y;
				pts[i].d=min(pts[i].d,val);
				val=query(1,n-pts[i].y+1)-pts[i].x-pts[i].y;
				pts[i].d=min(pts[i].d,val);
			}
		}
		v[z].clear();
		for(point p:pts)
			if(p.val==z)
				v[z].push_back(p);
	}
	int ans=INF;
	for(point p:v[m])
		ans=min(ans,p.d);
	fout<<ans+1<<'\n';
	return 0;
}
```

Vă recomandăm (ebooks)

Clubul Pythoniștilor

Limbajul **Python 3** este folosit la scară largă în prezent, fiind foarte popular, simplu de învățat și de utilizat. Spre deosebire de alte limbaje de programare, **Python 3** se înțelege foarte repede, *chiar de la început*.

Dă clic pe linkul de mai jos pentru mai multe detalii

https://ebooks.infobits.ro

Contact rapid

Telefon / WhatsApp
0727.731.947

Email
hello@infobits.ro