Zadaci sa rešenjima Srpska Informatička Olimpijada 2012.

zadatak: Žetoni

Mika igra jednu, ne tako poznatu, kockarsku igru žetoni. Igra se sastoji u sledećem: na početku, ispred njega se nalazi n gomila žetona u nizu, pri čemu je na i-toj gomili a_i žetona. Zatim Mika odigra određeni broj poteza, gde se pod jednim potezom podrazumeva izbor dva broja i i j ($i \leq j$) i dodavanje po jednog žetona svim gomilama između i-te i j-te, uključujući i ove dve gomile. Cilj igre je da, posle nekog broja poteza, na svim gomilama bude tačno k žetona.
Kada se igra završi, igrač dobija žetone na osnovu broja odigranih poteza - što manje poteza, to više žetona. Odredite koliko je najmanje poteza potrebno da bi Mika završio igru i možda ćete dobiti neke od Mikinih žetona.

Ulaz.
(Ulazni podaci se učitavaju iz datoteke **zetoni.in**.) U prvom redu ulazne datoteke nalaze se dva prirodna broja, n i k, koji predstavljaju, redom, broj gomila i traženi broj žetona na svakoj gomili na kraju igre ($1 \leq n \leq 10^6$, $1 \leq k \leq 10^9$). U narednom redu nalaze se n celih brojeva iz segmenta $[0, k]$ - opis gomila.

Izlaz.
(Izlazni podaci se ispisuju u datoteku **zetoni.out**) U prvom i jedinom redu izlazne datoteke ispisati jedan ceo broj - najmanji broj poteza potrebnih da se završi igra. Koristiti 64-bitni tip podataka.

Primer 1.

zetoni.in	zetoni.out
4 5 2 1 3 1	6

Objašnjenje. Ukoliko sa $[i, j]$ označimo dodavanje po jednog žetona gomilama i, $i + 1$, ..., j, onda nizom od 6 poteza [2, 4], [4, 4], [4, 4], [1, 4], [1, 2], [1, 2] dobijamo po 5 žetona na svakoj gomili, kao što je i trebalo. Ovo nije moguće postići u manje od 6 poteza.

Napomena.
U 30% test primera je $n \leq 10^3$.

fajl: zetoni.cpp

```
#include <cstdlib>
#include <cstdio>
#include <ctime>

const int MaxN = 1000100;

struct CoinPile
{
  int num, left, right;

  CoinPile(int n, int l, int r)
  {
    num = n; left = l; right = r;
  }
  CoinPile() {}
};

int n, k, x, pileNum, currIndex, nextIndex;
long long sol;
CoinPile coins[MaxN];

int main()
{
  freopen("zetoni.in", "r", stdin);
  freopen("zetoni.out", "w", stdout);

  scanf("%d%d", &n, &k);
  for (int i = 1; i <= n; i++)
  {
    scanf("%d", &x);
    coins[i] = CoinPile(x, i - 1, i + 1);
  }
  coins[0] = CoinPile(k + 1, 0, 1);
```

```
  coins[n + 1] = CoinPile(k + 1, n, n + 1);

  sol = -1;
  currIndex = 1;
  pileNum = n + 2;
  while (pileNum > 2)
  {
    if (coins[ coins[currIndex].left ].num <= coins[ coins[currIndex].right ].num)
      nextIndex = coins[currIndex].left;
    else
      nextIndex = coins[currIndex].right;

    if (coins[nextIndex].num >= coins[currIndex].num)
    {
      sol += (coins[nextIndex].num - coins[currIndex].num);
      pileNum--;
      coins[ coins[currIndex].left ].right = coins[currIndex].right;
      coins[ coins[currIndex].right ].left = coins[currIndex].left;
    }

    currIndex = nextIndex;
  }

  printf("%lld\n", sol);
  return 0;
}
```

fajl: zetoni.pas

```
const
  MaxN = 1000100;

var
  inFile, outFile : text;
  n, k, x, pileNum, currIndex, nextIndex, i : longint;
  sol : int64;
  num, left, right : array[0..MaxN] of longint;

BEGIN

  assign(inFile, 'zetoni.in');
  assign(outFile, 'zetoni.out');
  reset(inFile); rewrite(outFile);

  read(inFile, n, k);
  for i := 1 to n do begin
    read(inFile, num[i]);
    left[i] := i - 1;
    right[i] := i + 1;
  end;

  num[0] := k + 1; left[0] := 0; right[0] := 1;
  num[n + 1] := k + 1; left[n + 1] := n; right[n + 1] := n + 1;

  sol := -1;
  currIndex := 1;
  pileNum := n + 2;
  while (pileNum > 2) do begin

    if (num[ left[currIndex] ] <= num[ right[currIndex] ]) then
      nextIndex := left[currIndex]
    else
      nextIndex := right[currIndex];

    if (num[nextIndex] >= num[currIndex]) then begin
```

```
      sol := sol + (num[nextIndex] - num[currIndex]);
      pileNum := pileNum - 1;
      right[ left[currIndex] ] := right[currIndex];
      left[ right[currIndex] ] := left[currIndex];

    end;

    currIndex := nextIndex;

  end;

  writeln(outFile, sol);
  close(inFile);
  close(outFile);

END.
```

zadatak: Pijaca

Gazda Srbi šljiva nikad dosta i on redovno odlazi do pijace u susednom selu da razgleda nove sadnice za iste. Predstavljalo mu je problem to što je jako udaljena, međutim nedavno se otvorila nova pijaca u blizini i on je rešio da svakog jutra prošeta do nje ne bi li kupio nove sadnice za svoj šljivik.
Novootvorenu pijacu čini N tezgi koje se nalaze u redu jedna do druge. Na svakoj tezgi je moguće pronaći sadnice zajedno sa cenom za iste. Gazda Srba krene u obilazak počevši od neke tezge L i prođe redom svaku tezgu zaključno sa tezgom R, dok se zaustavlja samo na onim gde cena nije manja od A (jer zna da tog dana nisu kvalitetne) i na onim gde cena nije veća od B (jer su mu preskupe).
Gazda Srba uvek voli da isplanira ostatak dana, a dodatni problem mu predstavljaju cene koje variraju iz dana u dan, pa vas moli da mu napišete program koji bilo kada može odrediti broj tezgi na kojima će se zadržati.

Ulaz.

(Ulazni podaci se učitavaju iz datoteke **pijaca.in**.) U prvom redu ulazne datoteke se nalazi N ($1 \leq N \leq 50.000$), broj tezgi na pijaci. U narednom redu se nalazi niz X ($1 \leq N_i \leq 10^9$) od N brojeva koji predstavljalju početne cene sadnica na pijaci. Zatim, u sledećoj liniji, sledi Q ($1 \leq Q \leq 50.000$), broj upita koje trebate da odradite, i na kraju Q linija koje ih opisuju, oblika:
"1 *L R A B* - koji traži od vas da ispišete koliko ima tezgi od L do R ($1 \leq L \leq R \leq N$) sa cenom sadnica između A i B ($1 \leq A, B \leq 10^9$), uključujući A i B. "2 *I J*- u kojem se navodi da je nova cena sadnica na tezgi sa indeksom I ($1 \leq I \leq N$) jedanka J ($1 \leq J \leq 10^9$).

Izlaz.

(Izlazni podaci se ispisuju u datoteku **pijaca.out**) Za svaki upit tipa 1 redom iz ulaza u posebnoj liniji ispišite traženi broj tezgi.

Primer 1.

pijaca.in	**pijaca.out**
6 3 2 1 2 5 5 3 1 2 5 2 3 2 3 2 1 2 6 2 3	2 3

Objašnjenje. Odgovor na prvi upit je 2 jer se gazda Srba na svom putu zaustavlja kod tezgi 2 i 4, dok je odgovor na naredni upit tipa 1 jednak 3 jer je cena sadnice na tezgi broj 3 u međuvremenu poskupela na 2 i njegov naredni obilazak pijace će upotpuniti i ova tezga.

Napomena.

U 40% test primera, neće biti upita drugog tipa. U ovim primerima je Q neparno, a tamo gde ima i upita drugog tipa je Q parno.

fajl: pijaca.cpp

```
#include <stdio.h>
#include <algorithm>
#include <string>
using namespace std;
```

```
#define INF 2123456789

const int MaxN = 100005;

int a[MaxN], s[MaxN];
int n, q;

int upperBound(int left, int right, int x){
    while (left < right){
        int mid = (left + right) / 2;

        if (s[mid] <= x)
            left = mid + 1;
        else
            right = mid;
    }

    return s[left] > x ? left : left + 1;
}

int lowerBound(int left, int right, int x){
    while (left < right){
        int mid = (left + right) / 2;

        if (s[mid] < x)
            left = mid + 1;
        else
            right = mid;
    }

    return s[left] >= x ? left : left + 1;
}

int main(){
    FILE *fin = fopen("pijaca.in", "r");
    FILE *fout = fopen("pijaca.out", "w");

    fscanf(fin, "%d", &n);
    for (int i = 1; i <= n; i++){
        fscanf(fin, "%d", a+i);
        s[i] = a[i];
    }

    int k = 1;
    for (; k*k < n; k++);

    for (int i = n+1; i <= k*k; i++) a[i] = s[i] = INF;
    n = k*k;

    for (int i = 1; i <= k; i++) sort(s + (i - 1) * k + 1, s + i * k + 1);

    fscanf(fin, "%d", &q);
    while (q--){
        int command; fscanf(fin, "%d", &command);

        if (command == 1){
            int left, right, low, high; fscanf(fin, "%d%d%d%d", &left, &right, &low,
&high);

            int sol = 0;
            while (left <= right && right % k){
                sol += low <= a[right] && a[right] <= high;
                right--;
            }

            while (left <= right && left % k != 1){
```

```
                sol += low <= a[left] && a[left] <= high;
                left++;
            }

            while (left <= right){
                int upper = upperBound(right - k + 1, right, high);
                int lower = lowerBound(right - k + 1, right, low);
                //int upper = (int)(upper_bound(s + right - k + 1, s + right + 1, high) -
s);
                //int lower = (int)(lower_bound(s + right - k + 1, s + right + 1, low) -
s);

                sol += upper - lower;
                right -= k;
            }

            fprintf(fout, "%d\n", sol);
        }
        else {
            int idx, num; fscanf(fin, "%d%d", &idx, &num);
            int left = idx - (idx - 1) % k, right = left + k - 1;

            int tmp = a[idx];

            a[idx] = num;
            idx = lowerBound(left, right, tmp);
            //idx = (int)(lower_bound(s + left, s + right + 1, tmp) - s);
            s[idx] = num;

            while (left < idx && s[idx-1] > s[idx]){
                int p = s[idx-1]; s[idx-1] = s[idx]; s[idx] = p;
                idx--;
            }

            while (idx < right && s[idx] > s[idx+1]){
                int p = s[idx]; s[idx] = s[idx+1]; s[idx+1] = p;
                idx++;
            }
        }
    }

    fclose(fin); fclose(fout);
    return 0;
}
```

zadatak: Utaja poreza

Banja i Voža su se konačno obogatili. Vlasnici su korporacije Makrohard koja se u poslednje vreme pokazala kao neprikosnoveno najbolja u polju razvoja operativnih sistema i internet servisa. Makrohard je organizovan u vidu *n* poslovnica koje se nalaze u strogoj hijerarhiji. Poslovnica 1 je sedište korporacije i ona je, direktno ili indirektno (preko drugih poslovnica), nadređena svim ostalim. Svaka druga poslovnica ima tačno jednu direktno nadrešenu koja ima redni broj manji od nje. Vrednosti svih poslovnica su zadate u milionima dolara.
Na žalost, kako to obično biva, Banja i Voža su se posvašali oko dizajna zvanične šolje za kafu korporacije i odlučili da više ne mogu da posluju zajedno. Srećom, njihova dugogodišnja poznanica, a sada kontroverzni biznismen, Tana Rišović odlučila je da kupi Makrohard i tako pomogne Banji i Voži da se rastanu pre nego što dođe do većih incidenata. Međutim, svi znamo da milionske transakcije nikada nisu jednostavne i da ih, ukoliko ne želimo da veliki deo novca završi u rukama Poreske uprave, moramo pažljivo planirati.
Tanin računovođa je zaključio da, kako bi ukupan porez bio što manji, Tana svakog meseca treba da kupi tačno jednu poslovnicu Makroharda zajedno sa svim ostalim kojima je ona nadređena (direktno ili indirektno) a koje već ne poseduje, i pritom potroši najviše *k* miliona dolara. Vaš zadatak je da pomognete Tani tako što ćete izračunati minimalno trajanje ove transakcije.
Ulaz.

(Ulazni podaci se učitavaju iz datoteke **porez.in**.) U prvom redu ulazne datoteke nalaze se prirodni brojevi n ($1 \le n \le 100.000$) i k ($1 \le k \le 2^{30}$) - broj poslovnica firme Makrohard i količina novca koju Tana može da potroši svakog meseca. U sledećem redu nalazi se n prirodnih brojeva razdvojenih razmacima - qlanovi niza V , gde V_i predstavlja vrednost svake od poslovnica u milionima dolara ($V_i \le 2^{30}$). U trećem redu nalazi se n - 1 prirodan broj - članovi niza A, gde A_i predstavlja redni broj direktno nadređene poslovnice za poslovnicu $i + 1$ ($A_i \le i$).

Izlaz.

(Izlazni podaci se ispisuju u datoteku **porez.out**) U prvom i jedinom redu izlazne datoteke ispisati jedan ceo broj - najmanji broj mesečnih transakcija potreban da bi se prodaja Makroharda okončala.

Primer 1.

porez.in	**porez.out**
6 20 7 6 7 5 6 4 1 1 2 4 4	2

Objašnjenje. Tana će u prvom mesecu kupiti poslovnice 4,5 i 6, a u drugom 1,2 i 3.

Napomena. Transakcija će uvek moći da se obavi.

fajl: porez.cpp

```
#include <stdio.h>
#include <algorithm>
#include <vector>
using namespace std;
int n,k;
int mint[100000]; //najmanji broj kupovina za podstablo sa korenom i
int minr[100000]; //minimalna vrednost pri poslednjoj kupovini za mint[i]
int v[100000];
int nad[100000];
vector <int> potomak[100000]; //potomci cvora i
bool cmp (int a,int b) { return minr[a]<minr[b]; }
int main ()
{
    freopen("porez.in","r",stdin);
    freopen("porez.out","w",stdout);
    scanf("%d %d",&n,&k);
    for (int i=0;i<n;i++) scanf("%d",&v[i]);
    for (int i=1;i<n;i++) {scanf("%d",&nad[i]);nad[i]--;potomak[nad[i]].push_back(i);}
    for (int i=n-1;i>=0;i--)
    {
        int t=1;
        int r=v[i];
        sort(potomak[i].begin(),potomak[i].end(),cmp);
        for (int j=0;j<potomak[i].size();j++)
        {
            t+=mint[potomak[i][j]];
            if (r+minr[potomak[i][j]] <= k)
            {
                r+=minr[potomak[i][j]];
                t--;
            }
        }
        mint[i]=t;
        minr[i]=r;
    }
    printf("%d\n",mint[0]);
}
```

zadatak: Muzej

Ispred muzeja programerskih nauka u Nišu je velika gužva - već se napravio red od n posetilaca. Među osobama u redu ima onih koji nemaju nikakve popuste u muzeju (**'b'** tip osoba) i onih koji imaju 50% popusta jer su rešavali Problem Meseca (**'a'** tip osoba).

Laza radi kao organizator u muzeju i njegov posao je da posetioce, u redosledu dolaska, raspoređuje u grupe, gde je grupa skup nekoliko uzastopnih osoba u redu. Svaka osoba iz reda mora pripadati taqno jednoj grupi i nijedna grupa ne sme biti prazna. Takođe, zbog politike muzeja, **u svakoj grupi, broj osoba tipa 'a' mora biti veći ili jednak od broja osoba tipa 'b'**. Posle formiranja, jedna po jedna grupa, redom od početka reda, provodi 1 sat u obilasku muzeja dok preostale grupe čekaju.
Iako obiqno radi svoj posao vrlo savesno, Laza je zapazio da je *m*-ta osoba u redu zapravo član redakcije Problema Meseca koji mu jednom nije priznao zadatak i zbog toga je odlučio da sada napravi izuzetak. On želi da podeli posetioce u grupe tako da pomenuti član redakcije čeka što duže pre nego što na njegovu grupu dođe red. Pomozite mu u tome.

Ulaz.

(Ulazni podaci se učitavaju iz datoteke **muzej.in**.) U prvom redu ulazne datoteke nalaze se dva prirodana broja *n* i *m*, redom, broj osoba koje čekaju u redu i pozicija pomenutog člana redakcije ($1 \leq m \leq n \leq 300.000$). U narednom redu nalazi se string dužine *n* sastavljen isključivo od slova *a* i *b* - opis reda. Početak stringa je početak reda dok slova predstavljaju odgovarajuće tipove osoba. Ulazni podaci će biti takvi da će Laza uvek moći da izvrši neki raspored posetilaca u grupe.

Izlaz.

(Izlazni podaci se ispisuju u datoteku **muzej.out**) U prvom i jedinom redu izlazne datoteke ispisati jedan ceo broj - broj sati koje će morati da čeka *m*-ta osoba u redu, pri najnepovoljnijoj podeli, pre nego što dođe red na njegovu grupu.

Primer 1.

muzej.in	**muzej.out**
10 8 aabaabbbaa	4

Objašnjenje. Ukoliko red podelimo na 5 grupa: *a* | *ab* | *a* | *ab* | *bbaa*, osma osoba (podvučena je) će morati da čeka prve 4 grupe, tj. čekaće 4 sata. Ne postoji podela u kojoj osma osoba u redu čeka duže.

Napomena.

U 30% test primera je $n \leq 3.000$, dok je u 60% test primera $n = m$, tj. član redakcije je poslednja osoba u redu.

fajl: muzej.cpp

```
#include <cstdlib>
#include <cstdio>

const int MaxN = 1000100;

int n, m, sol;
char s[MaxN];
int p[MaxN], d[MaxN], rightmost[MaxN];

int main()
{
  freopen("muzej.in", "r", stdin);
  freopen("muzej.out", "w", stdout);

  scanf("%d%d", &n, &m);
  scanf("%s", s);

  p[0] = 0;
  for (int i = 1; i <= n; i++)
    if (s[i - 1] == 'a')
      p[i] = p[i - 1] + 1;
    else
      p[i] = p[i - 1] - 1;

  for (int i = 0; i <= n; i++)
    rightmost[i] = 0;

  d[0] = 0;
  for (int i = 1; i <= n; i++)
  {
    if (p[i] >= 0)
    {
```

```
      d[i] = d[ rightmost[ p[i] ] ] + 1;
      if (p[i - 1] < p[i] && d[i - 1] + 1 > d[i])
        d[i] = d[i - 1] + 1;

      rightmost[ p[i] ] = i;
    }
    else
    {
      d[i] = -1;
    }
  }

  sol = 0;
  for (int i = 0; i < m; i++)
    if (d[i] > sol && p[n] - p[i] >= 0) sol = d[i];

  printf("%d\n", sol);
  return 0;
}
```

fajl: muzej.pas

```
const
  MaxN = 1000100;

var
  inFile, outFile : text;
  n, m, sol, i : longint;
  s : array[0..MaxN] of char;
  p, d, rightmost : array[0..MaxN] of longint;

BEGIN

  assign(inFile, 'muzej.in');
  assign(outFile, 'muzej.out');
  reset(inFile); rewrite(outFile);

  readln(inFile, n, m);
  for i := 1 to n do
    read(inFile, s[i]);

  p[0] := 0;
  for i := 1 to n do
    if (s[i] = 'a') then
      p[i] := p[i - 1] + 1
    else
      p[i] := p[i - 1] - 1;

  for i := 0 to n do
    rightmost[i] := 0;

  d[0] := 0;
  for i := 1 to n do begin

    if (p[i] >= 0) then begin

      d[i] := d[ rightmost[ p[i] ] ] + 1;
      if ((p[i - 1] < p[i]) AND (d[i - 1] + 1 > d[i])) then
        d[i] := d[i - 1] + 1;

      rightmost[ p[i] ] := i;

    end
    else
      d[i] := -1;
  end;
```

```
  sol := 0;
  for i := 0 to m - 1 do
    if ((d[i] > sol) AND (p[n] - p[i] >= 0)) then
      sol := d[i];

  writeln(outFile, sol);
  close(inFile);
  close(outFile);

END.
```

zadatak: Pentranje

Ako se dobro sećate, gazda Srba ima voćnjak šljiva u srcu Šumadije. Kako je Srba veliki perfekcionista, on svoj šljivik održava uvek u formi kvadrata, tako da u svakom od ukupno *N* redova šljivika bude tačno po *N* stabala šljiva. Svako stablo ima svoju visinu, koju gazda Srba uredno kontroliše.
Kada se Srba popne na neku šljivu, on vidi vrhove svih šljiva u voćnjaku koje imaju manju visinu od one na koju se popeo (vrhove ostalih šljiva ne vidi). Njega zanima, kada se popne na neku šljivu, koji je najdalji vrh koji on vidi. Srba rastojanje između dve šljive meri kao zbir apsolutnih razlika redova i kolona u kojima se nalaze. Vremenom će se i visine nekih šljiva promeniti (gazda Srba može da ih poseče ili kalemi sa ciljem da bolje rode sledeće godine), ali će vas blagovremeno obavestiti o svim promenama visina.
Formalnije, gazda Srba će vam dostaviti *Q* informacija. Informacija može da bude tipa "1 *X Y*" što označava da se gazda Srba popeo na šljivu koja se nalazi u redu *X* i koloni *Y* , na ovu informaciju vi njemu treba da odgovorite koliko je udaljena najdalja šljiva čiji vrh on trenutno vidi; drugi tip informacije je "2 *X Y V*", što označava da je šljiva koja se nalazi u redu *X* i koloni *Y* promenila visinu na *V*.
Ulaz.
(Ulazni podaci se učitavaju iz datoteke **pentranje.in**.) U ulaznoj datoteci se u prvom redu nalazi broj *N* ($1 \leq N \leq 1000$), broj redova i kolona Srbinog voćnjaka. U sledećih *N* redova se nalaze po *N* brojeva iz intervala $[1, 10^9]$ koji predstavljaju početne visine svake šljive. U sledećem redu se nalazi broj *Q* ($1 \leq Q \leq 500.000$), broj Srbinih informacija. U sledećih *Q* redova se nalaze gore opisane informacije, u svakom redu po jedna. ($1 \leq X,Y \leq N$, $1 \leq V \leq 10^9$).
Izlaz.
(Izlazni podaci se ispisuju u datoteku **pentranje.out**) Za svaku informaciju prvog tipa ispisati udaljenost najudaljenije šljive čiji vrh gazda Srba vidi kada se popne na tu šljivu. Ukoliko ne postoji šljiva manja od one na koju se on popeo, ispisati -1.
Primer 1.

pentranje.in	pentranje.out
5 6 7 8 2 3 4 8 2 4 7 9 8 2 8 3 1 7 1 8 2 2 5 5 3 9 7 1 5 1 2 5 1 3 1 5 1 2 2 5 1 1 5 1 1 2 5 1 3 5	3 5 7 -1 5

Objašnjenje. Gazda Srba vam daje prvu informaciju da se popeo na šljivu (5, 1). On sada vidi vrhove šljiva (4, 1) i (4, 3). Šljiva (4, 1) se nalazi na udaljenosti 1 dok se šljiva (4, 3) nalazi na udaljenosti 3, pa je rezultat 3. Druga informacija je da je šljiva (5, 1) promenila visinu na 3. Treća informacija takođe kaže da se Srba popeo na šljivu (5, 1), ali sada on vidi vrhove šljiva (2, 3), (3, 3), (4, 1), (4, 3), (4, 5), od kojih su najudaljeniji (2, 3) i (4, 5) i nalaze se na udaljenosti 5. Za petu informaciju najudaljenija šljiva je (2, 5), za šestu nema šljive koja ima visinu manju od 1 pa je odgovor -1, a za poslednju informaciju najudaljenija manja šljiva je (4, 1).

Napomena.
U 50% test primera, Srba vam nikad neće dati informaciju drugog tipa. U ovim primerima je *Q* neparno, a tamo gde ima i informacija drugog tipa je *Q* parno.

fajl: pentranje.cpp

```
#include <vector>
#include <list>
#include <map>
#include <set>
#include <deque>
#include <stack>
#include <bitset>
#include <algorithm>
#include <functional>
#include <numeric>
#include <utility>
#include <sstream>
#include <iostream>
#include <iomanip>
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <queue>
#include <cstring>

using namespace std;

#define sz size()
#define pb push_back
#define len length()
#define clr clear()
#define FOR(i,a,b) for(i=a;i<b;i++)
#define FORR(i,n) for(i=0;i<n;i++)
#define is_digit(c) ('0'<=(c) && (c)<='9')

#define eps 0.0000001
#define PI  3.14159265359
#define inf 1999888777

int n,a[1005][1005],where_glavna[1005][1005],where_sporedna[1005][1005];


struct polje {
    int x,y;
} pos_glavna[2000555],pos_sporedna[2000555];

int dist(int x1, int y1, int x2, int y2) {
    return (abs(x1-x2) + abs(y1-y2));
}

class segmentno {

 public:
    int brc,m[3500555];
    bool glavna;

    void init(bool g) {

        int i;

        brc = 1;
        while (brc < n*n) brc *= 2;
```

```
        for(i=brc; i<2*brc; i++) {
            m[i] = inf;
        }
        for(i=brc-1; i>0; i--) {
            m[i] = inf;
        }

        glavna = g;
    }

    polje nadji_najblizi_levo(int v) {

        int x;

        x = 1;
        while (x < brc) {
            if (m[x*2] < v) x = x*2; else x = x*2 + 1;
        }

        if (glavna) {
            return pos_glavna[x-brc+1];
        } else {
            return pos_sporedna[x-brc+1];
        }
    }

    polje nadji_najblizi_desno(int v) {

        int x;

        x = 1;
        while (x < brc) {
            if (m[x*2+1] < v) x = x*2 + 1; else x = x*2;
        }

        if (glavna) {
            return pos_glavna[x-brc+1];
        } else {
            return pos_sporedna[x-brc+1];
        }
    }

    polje nadji_najdalji_manji(int x, int y) {

        polje p1,p2;

        p1 = nadji_najblizi_levo(a[x][y]);
        p2 = nadji_najblizi_desno(a[x][y]);

        return (dist(x,y,p1.x,p1.y) > dist(x,y,p2.x,p2.y)) ? p1 : p2;
    }

    void ubaci(int k, int v) {

        int x;

        x = brc + k - 1;
        m[x] = v;
        x /= 2;
        while( x > 0 ) {
            m[x] = min(m[2*x], m[2*x+1]);
            x /= 2;
        }
    }
};

void init_where_glavna() {
```

```
    int i,j,pi,pj,br;

    br=0;
    for(i=0; i<n; i++) {
        pi=i; pj=0;
        while(pi >= 0) {
            br++;
            where_glavna[pi][pj] = br;
            pos_glavna[br].x = pi;
            pos_glavna[br].y = pj;
            pi--;
            pj++;
        }
    }
    for(j=1; j<n; j++) {
        pi=n-1; pj=j;
        while(pj < n) {
            br++;
            where_glavna[pi][pj] = br;
            pos_glavna[br].x = pi;
            pos_glavna[br].y = pj;
            pi--;
            pj++;
        }
    }
}

void init_where_sporedna() {

    int i,j,pi,pj,br;

    br=0;
    for(i=n-1; i>=0; i--) {
        pi=i; pj=0;
        while(pi < n) {
            br++;
            where_sporedna[pi][pj] = br;
            pos_sporedna[br].x = pi;
            pos_sporedna[br].y = pj;
            pi++;
            pj++;
        }
    }
    for(j=1; j<n; j++) {
        pi=0; pj=j;
        while(pj < n) {
            br++;
            where_sporedna[pi][pj] = br;
            pos_sporedna[br].x = pi;
            pos_sporedna[br].y = pj;
            pi++;
            pj++;
        }
    }
}

segmentno seg_glavna, seg_sporedna;

int main() {

    FILE *fin = fopen("pentranje.in", "r"), *fout = fopen("pentranje.out", "w");

    int i,j,x,y,v,q,up;
    polje p1,p2;

    fscanf(fin, "%d", &n);
```

```
    //scanf("%d", &n);

    init_where_glavna();
    init_where_sporedna();

    seg_glavna.init(true);
    seg_sporedna.init(false);

    for(i=0; i<n; i++) {
        for(j=0; j<n; j++) {
            fscanf(fin, "%d", &a[i][j]);

            seg_glavna.ubaci(where_glavna[i][j], a[i][j]);
            seg_sporedna.ubaci(where_sporedna[i][j], a[i][j]);
        }
    }

    fscanf(fin,"%d", &q);
    for(i=0; i<q; i++) {
        fscanf(fin,"%d", &up);

        if (up == 2) {
            fscanf(fin,"%d%d%d", &x, &y, &v);
            x--; y--;
            a[x][y] = v;
            seg_glavna.ubaci(where_glavna[x][y], v);
            seg_sporedna.ubaci(where_sporedna[x][y], v);
        } else {
            fscanf(fin, "%d%d", &x, &y);
            x--; y--;
            if (seg_glavna.m[1] >= a[x][y]) {
                fprintf(fout,"-1\n");
            } else {
                p1 = seg_glavna.nadji_najdalji_manji(x,y);
                p2 = seg_sporedna.nadji_najdalji_manji(x,y);
                fprintf(fout,"%d\n", max( dist(x,y,p1.x,p1.y), dist(x,y,p2.x,p2.y)));
            }

        }
    }

    fclose(fin); fclose(fout);

    return 0;
}
```

zadatak: Flojd

Ko je ona tvrda glava, što do podne uvek spava? Ko je dasa u ekstazi, kada preko stotke gazi? Naravno, to je Flojd. Flojd je mudar, hrabar i ludo vozi. Međutim, čak i takvom reli asu je ponekad potrebna pomoć. Organizatori predstojeće reli trke su odlučili da uvedu nova pravila. Na stazu su postavili N zastavica numerisanih brojevima od 1 do N, tako da i-ta zastavica ima koordinate (X_i, Y_i). Kao i svaki pravi reli, i ovaj naš se odvija na neuređenom, zemljanom terenu, na kome se tragovi točkova savršeno vide. Pravila trke su sledeća: vozač treba da izabere redosled zastavica z_1, ..., z_N, i potom pravolinijski ide od zastavice numerisane brojem z_1 do z_2, od z_2 do z_3, i tako sve dok ne stigne do zastavice z_N. Naravno, kako bi se umešnost vozača što više stavila na ispit, postoje i dodatna pravila koja donose dodatne poene. Naime, iz početne zastavice vozač sme da ide samo severno, odnosno ka zastavici koja ima veću y koordinatu. Potom sme da ide samo južno, odnosno ka zastavici koja ima manju y koordinatu, i tako dalje. Drugim reqima, treba da važi $y_{z1} < y_{z2} > y_{z3} < y_{z4} > ....$ Takođe, nije dozvoljeno da trag koji točkovi ostavljaju ima samopresecanja, osim u tačkama na kojima se nalaze zastavice, i naravno, potrebno je obići svaku zastavicu tačno jednom. Pomozite Flojdu da nađe takvu putanju i time osvoji sve moguće poene i srca publike!

Ulaz.

(Ulazni podaci se učitavaju iz datoteke **flojd.in**.) U ulaznoj datoteci se u prvom redu nalazi broj N ($1 \leq N \leq 5000$), broj zastavica. U sledećih N redova se nalaze po dva cela broja iz intervala $[1, 10^7]$, pri čemu i-ti red sadrži (x_i, y_i), koordinate i-te zastavice. Nikoje tri tačke nisu kolinearne i ne postoje dve tačke sa istom y koordinatom.

Izlaz.

(Izlazni podaci se ispisuju u datoteku **flojd.out**) Ukoliko postoji putanja koja zadovoljava kriterijume, ispisati redne brojeve zastavice u redosledu u kojem ih treba obići, po jednu u svakom redu. Ukoliko postoji više mogućih putanja, štampati bilo koju. Ukoliko takva putanja ne postoji, ispisati -1.

Primer 1.

flojd.in	flojd.out
3 1 1 5 5 2 3	1 2 3

Napomena.

U 75% test primera ima najviše 3000 zastavica.

fajl: flojd.cpp

```
#include <iostream>
#include <cstdio>
using namespace std;

const char inFileName[] = "flojd.in";
const char outFileName[] = "flojd.out";

const int MaxN = 5001;

int n;
long x[MaxN], y[MaxN];

int next[MaxN], prev[MaxN];
bool used[MaxN];

bool right(int a, int b, int c) {
  long long A = y[b] - y[a], B = x[a] - x[b], C = A * x[a] + B * y[a];
  return A * x[c] + B * y[c] - C > 0;
}

int find_right(int a) {
  int r = -1;

  for (int i = 0; i < n; i++) {
    if (i == a || used[i]) continue;
    else {
      if (r == -1) r = i;
      else if (right(a, r, i)) r = i;
    }
  }
  return r;
}

int convex_hull() {
  long ysmall = y[0];
  int ind = 0;
  for (int i = 1; i < n; i++) {
    if (y[i] < ysmall) {
      ysmall = y[i];
      ind = i;
    }
  }

  int start = ind;
  next[ind] = ind;
```

```
  prev[ind] = ind;
  bool done = false;
  while (!done) {
    int nextInd = find_right(ind);
    prev[nextInd] = ind;
    next[ind] = nextInd;
    ind = nextInd;
    if (nextInd == start)
      done = true;
  }

  return start;
}

void delete_point(int p) {
  int s = prev[p], q = next[p];
  while (s != q) {
    int nextS = find_right(s);
    next[s] = nextS;
    prev[nextS] = s;
    s = nextS;
  }
}

int main() {
  freopen(inFileName, "r", stdin);
  freopen(outFileName, "w", stdout);

scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    scanf("%ld %ld", &x[i], &y[i]);
    used[i] = false;
  }

  int currentPoint = convex_hull();
  bool pickHigher = true;
  for (int i = 0; i < n - 3; i++) {
    printf("%d\n", currentPoint+1);
    int c1 = prev[currentPoint], c2 = next[currentPoint];
    used[currentPoint] = true;
    delete_point(currentPoint);
    if (pickHigher) {
      if (y[c1] > y[c2]) currentPoint = c1;
      else currentPoint = c2;
    } else {
      if (y[c1] < y[c2]) currentPoint = c1;
      else currentPoint = c2;
    }
    pickHigher = !pickHigher;
  }

  int c1 = prev[currentPoint], c2 = next[currentPoint];
  if (pickHigher) {
    if (y[c1] < y[c2]) swap(c1, c2);
  } else {
    if (y[c1] > y[c2]) swap(c1, c2);
  }
  printf("%d\n%d\n%d\n", currentPoint+1, c1+1, c2+1);
  return 0;
}
```