Zadaci sa rešenjima Srpska Informatička Olimpijada 2008.

zadatak: Moćni rendžeri

Moćni rendžeri su superheroji koji brane Zemlju od napada zlih sila predvođenih kraljicom Ritom i gospodarom Zedom. U stalnoj borbi protiv zla pomažu im njihovi roboti zvani zordovi. Ipak, zordovi nemaju neiscrpan izvor energije, već se moraju dopunjavati s vremena na vreme. U jednoj bici angažovan je određen broj zordova: neki od njih se bore (i tako troše energiju), dok se drugi dopunjavaju. Po završenoj bici, zordovi koji su se borili potrošili su svu energiju, i ne mogu se boriti u narednim bitkama sve dok se ne dopune, a zordovi koji su se dopunjavali ostaju puni i spremni da se bore kad zatreba. S obzirom na to što zle sile u nameri da porobe Zemlju stalno šalju sve moćnija i moćnija čudovišta, u svakoj bici potrebno je angažovati sve više i više zordova: preciznije, u *i*-toj po redu bici potrebno je angažovati tačno *i* zordova (od kojih se neki bore, a ostali se dopunjavaju).
Zordovi se dopunjavaju pomoću solartomske energije. Vođa Rendžera, Zordon, mora da vodi preciznu evidenciju koliko zordova se dopunjava u kojoj bici, kako bi znao da svom pomoćniku, robotu Alfi, kaže koliko solartomske energije treba da kupi u dućanu na uglu. Ipak, vremena su teška, te i Zordon kupuje, narodski rečeno, 'na recku'; drugim rečima, nakon što Rendžeri uspešno odbrane Zemlju i proteraju zle sile iz vasione (za šta im treba mnogo bitaka), Zordon izmiruje račune s dućanom.
Ovaj put posao Rendžera bio je zaista težak: od *n* zordova koje poseduju (gde je *n* neparan broj, da zlikovci ne bi mogli podeliti rendžere na dva jednaka dela i napasti svaki deo ponaosob), u poslednjoj bici, *n*-toj po redu, morali su da angažuju sve! Srećom, u toj bici izvojevali su konačnu pobedu i zauvek raskrstili sa zlim silama, pa neće biti više napada (nećemo ni pomišljati šta bi bilo da su se pokvarenjaci uspeli zadržati još malo i poslati još moćnije čudovište, koje bi zahtevalo *n+1* zordova; samo jedno je sigurno: u tom slučaju niko od vas ne bi prošao na IOI, jer ne bi ni bilo IOI, jer ne bi bilo ni Zemlje, ali ostavimo sitnice po strani).
U svoj toj golgoti Zordon je negde zaturio ceduljče na koje je beležio koliko se zordova dopunjavalo u kojoj bici. Ostalo mu je zabeleženo samo koji od zordova su na početku (pre svih bitaka) bili puni a koji prazni, i još može uočiti da su sada, posle svega, svi zordovi istog stanja (tj. ili su svi puni, ili su svi prazni). Vlasnik dućana je jako ljut i preti da će mu zapleniti neke stvari iz Komandnog centra, te vas Zordon moli za pomoć: odredite koliko je zordova u kojoj bici bilo dopunjavano.

Ulaz:
(Ulazni podaci se nalaze u datoteci **rendzeri.in**) U prvom redu ulazne datoteke nalazi se prirodan, neparan broj *n* ($1 \leq n < 10^6$), broj zordova koje Rendžeri poseduju. U narednom redu nalazi se niz znakova '+' i '-', dužine *n*, gde je *i*-ti znak jednak '+' ako je *i*-ti zord bio pun pre dolaska zlikovaca, a '-' ako je bio prazan.

Izlaz:
(Izlazne podatke upisati u datoteku **rendzeri.out**) U *i*-tom redu izlazne datoteke treba upisati koliko se zordova dopunjavalo tokom *i*-te bitke. Ako se zadatak može rešiti na više načina, ispisati bilo koji.

Primer 1:

rendzeri.in	**rendzeri.out**
5	1
-+---	2
	0
	4
	0

Objašnjenje.
U pitanju je prvobitna postava Rendžera: Zak, Kimberli, Bili, Trini i Džejson. Na slici je prikazano kako su se oni borili: bledom nijansom prikazani su prazni zordovi, a jakim tonom puni. Vidimo da su na kraju svi zordovi istog stanja (u ovom slučaju prazni), što se poklapa s uslovom zadatka. Takođe možemo videti da u *i*-toj bici učestvuje tačno *i* zordova (podsećamo: neki se bore i time troše energiju, a ostatak od tih *i* se dopunjuje), kao što je rečeno.

Primer 2:

rendzeri.in	**rendzeri.out**

```
7              0
+++++++        0
               0
               3
               4
               0
               7
```

Rešenje

Najpre opisujemo proceduru pomoću koje, za neparan broj *k*, postižemo da *k* datih zordova koji su inicijalno istog stanja nakon *k* bitaka i dalje budu međusobno istog stanja. Poređajmo tih *k* zordova ukrug i angažujmo ih redom (ciklično), svaki put nastavljajući tamo gde smo prethodno stali. Kako imamo *k* zordova i ukupno $1 + 2 + 3 + ... + k = k*(k + 1)/2$ angažmana, konstatujući da je ovaj broj deljiv sa *k* (jer je *k* neparno), zaključujemo da ćemo stati upravo nakon što obiđemo krug ceo broj puta, pa će zaista na kraju svi zordovi biti istog stanja.
Pogledajmo sada kako se ovo primenjuje na naš problem. Neka je na početku *l* praznih i $n - l$ punih zordova (moguće je i $l = 0$), i uzmimo $l < n - l$ (u drugom slučaju rezonujemo potpuno isto). Izdvojmo svih *l* praznih zordova i još *l* punih. Preostalih $n - 2l$ zordova jesu svi puni, i njih angažujmo tokom prvih $n - 2l$ bitaka ciklično (tj. prema goreopisanoj proceduri, jer je $n - 2l$ neparan broj), posle čega će oni ostati međusobno istog stanja. U svakoj narednoj bici, recimo *i*-toj (gde je $n > n - 2l$), angažujmo sve zordove koji su bili angažovani i u prethodnoj bici, $(i - 1)$-oj po redu, i njima priključimo jednog od izdvojenih koji je s njima istog stanja. Ovim postupkom postižemo da posle *i*-te bitke uvek imamo (bar) *i* zordova istog stanja, te ćemo posle *n*-te bitke imati *n* zordova istog stanja, što je i traženo.

fajl: rendzeri.pas

```
var n,i,l,d:longint;
    c:char;
    f:text;
procedure ciklicno(n,l:longint);
var ispred:boolean;
begin
 ispred:=true;
 for i:=1 to n do
   begin
    if ispred then if i<l then begin
                                writeln(f,i);
                                l:=l-i;
                               end
                           else begin
                                writeln(f,l);
                                l:=i-l;
                                ispred:=false;
                               end
              else if i<n-l then begin
                                  writeln(f,0);
                                  l:=l+i;
                                 end
                            else begin
                                  writeln(f,i+l-n);
                                  l:=2*n-i-l;
                                  ispred:=true;
                                 end;
   end;
end;
begin
 assign(f,'rendzeri.in');
 reset(f);
 readln(f,n);
 for i:=1 to n do
   begin
```

```
    read(f,c);
    if c='-' then inc(l);
   end;
 close(f);
 assign(f,'rendzeri.out');
 rewrite(f);
 if l<n-l then d:=l
          else d:=n-l;
 ciklicno(n-2*d,l-d);
 for i:=d-1 downto 0 do
   if ((l-i) mod 2)=((n-2*i+1) mod 4 div 2) then begin
                                                  writeln(f,0);
                                                  writeln(f,n-2*i);
                                                 end
                                           else begin
                                                  writeln(f,n-2*i-1);
                                                  writeln(f,0);
                                                 end;
 close(f);
end.
```

zadatak: Stabla

Odrediti broj minimalnih pokrivajućih stabala datog povezanog težinskog grafa u kome se ni jedna težina grane ne ponavlja više od 4 puta.

Ulaz:

(Ulazni podaci se nalaze u datoteci **stabla.in**) U prvom redu ulaza su zapisani brojevi N i M ($1 \leq N,M \leq 5\text{x}10^4$), broj čvorova i broj grana datog grafa. U svakom od narednih M redova zapisana su tri cela broja A, B i W ($1 \leq A,B \leq N$, $1 \leq W \leq 2^{30}$), koji označavaju da između čvorova A i B postoji grana težine W.

Izlaz:

(Izlazne podatke upisati u datoteku **stabla.out**) U izlaznu datoteku ispisati ostatak traženog broja minimalnih pokrivajućih stabala datog grafa koji se dobija pri deljenju sa 1000003.

Primer 1:

stabla.in	**stabla.out**
3 4 1 2 6 1 2 6 2 3 6 3 1 6 3 3 8	5

Objašnjenje.

Sve grane su iste težine, između čvorova 1 i 2 postoje dve grane, a postoji i grana kojoj su oba kraja u čvoru 3.

Primer 2:

stabla.in	**stabla.out**
6 8 1 2 1 2 3 2 3 4 3 4 5 3 5 6 2 6 1 1 2 6 1	

3 5 3

fajl: stabla.cpp

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

const long MaxN = 100001;

const long MOD   = 1000003;


struct Edge{
  long a, b;
  long w;

  Edge(){}
  Edge(long _a, long _b, long _w):a(_a), b(_b), w(_w){}

  friend bool operator<(const Edge &a, const Edge &b){
    return a.w < b.w;
  }
};


Edge e[MaxN];
long n, m;

long rez;

bool root[MaxN];
long otac[MaxN];


void init_disjoint(){
  for (long i = 0; i < MaxN; i++){
    root[i] = true;
    otac[i] = 1;
  }
}

long group(long x){
  long tmp = x, tmp1;
  while (!root[x]) x = otac[x];
  while (!root[tmp]){
    tmp1 = otac[tmp];
    otac[tmp] = x;
    tmp = tmp1;
  }
  return x;
}

void merge(long _x, long _y){
  long x = group(_x), y = group(_y);
  if (x == y) return;

  if (otac[x] > otac[y]){
    otac[x] += otac[y];
    otac[y] = x;
    root[y] = false;
  }
  else{
    otac[y] += otac[x];
    otac[x] = y;
```

```
    root[x] = false;
  }
}

void count2(long *a, long *b, long idBr){
  if (idBr == 2) rez *= 2;
}

void count3(long *a, long *b, long idBr){
  if (idBr == 2) rez *= 3;
  else if (idBr == 3){
    if ((a[0] == a[1] && b[0] == b[1]) || (a[0] == a[2] && b[0] == b[2]) || (a[1] == a[2]
&& b[1] == b[2])) rez *= 2;
    else rez *= 3;
  }
  else if (idBr == 4){
    if ((a[0] == a[1] && b[0] == b[1]) || (a[0] == a[2] && b[0] == b[2]) || (a[1] == a[2]
&& b[1] == b[2])) rez *= 2;
  }
}

void count4(long *a, long *b, long idBr){
  bool dva = false;
  for (int i = 0; !dva && i < 4; i++) for (int j = i + 1; !dva && j < 4; j++)
    if (a[i] == a[j] && b[i] == b[j])  dva = true;

  bool usamljenaGrana = false;
  for (int i = 0; !usamljenaGrana && i < 4; i++){
    usamljenaGrana = true;
    for (int j = 0; usamljenaGrana && j < 4; j++){
      if (i == j) continue;
      if (a[i] == a[j] || a[i] == b[j] || b[i] == a[j] || b[i] == b[j]) usamljenaGrana =
false;
    }
  }

  int stepen[8];
  for (int i = 0; i < 8; i++) stepen[i] = 0;
  for (int i = 0; i < 4; i++){
    stepen[a[i]-1]++;
    stepen[b[i]-1]++;
  }
  sort(&stepen[0], &stepen[idBr]);

  if (idBr == 2) rez *= 4;
  else if (idBr == 3){
    if (stepen[0] == 2 && stepen[1] == 2 && stepen[2] == 4) rez *= 4;
    else if (stepen[0] == 1 && stepen[1] == 3 && stepen[2] == 4) rez *= 3;
    else rez *= 5;
  }
  else if (idBr == 4){
    if (stepen[0] == 2 && stepen[1] == 2 && stepen[2] == 2 && stepen[3] == 2) rez *= 4;
    else if (dva && usamljenaGrana) rez *= 3;
    else{
      if (dva) rez *= 2;
      else if (stepen[0] == 1 && stepen[1] == 2 && stepen[2] == 2 && stepen[3] == 3) rez *=
3;
    }
  }
  else if (idBr == 5){
    if (dva) rez *= 2;
    else if (usamljenaGrana) rez *= 3;
  }
  else if (idBr == 6){
    if (dva) rez *= 2;
  }
}
```

```
void count(long *paket, long br){
  if (br <= 1) return;
  else{
    map<long, long> mapa;
    long idBr = 1;
    long a[4], b[4];

    for (int i = 0; i < br; i++){
      long g1 = group(e[paket[i]].a);
      long g2 = group(e[paket[i]].b);
      if (mapa[g1] == 0) mapa[g1] = idBr++;
      if (mapa[g2] == 0) mapa[g2] = idBr++;
      a[i] = mapa[g1]; b[i] = mapa[g2];
      if (a[i] > b[i]) swap(a[i], b[i]);
    }

    idBr--;

    if (br == 4) count4(a, b, idBr);
    else if (br == 3) count3(a, b, idBr);
    else if (br == 2) count2(a, b, idBr);

    rez %= MOD;
  }
}

void Kruskal(){
  init_disjoint();
  sort(&e[0], &e[m]);

  rez = 1; // pretpostavljam da je graf povezan

  long last = e[0].w;

  long nextGroup[4];
  long br = 0;

  nextGroup[br++] = 0;

  for (long i = 1; i < m; i++){
    if (e[i].w != last){
      count(nextGroup, br);

      for (int j = 0; j < br; j++) merge(e[nextGroup[j]].a, e[nextGroup[j]].b);

      br = 0;
      last = e[i].w;

    }

    long g1 = group(e[i].a);
    long g2 = group(e[i].b);

    if (g1 != g2) nextGroup[br++] = i;
  }

  count(nextGroup, br);
  for (int j = 0; j < br; j++) merge(e[nextGroup[j]].a, e[nextGroup[j]].b);
}

void init(){
  FILE *f;
  f = fopen("stabla.in", "r");
  fscanf(f,"%ld %ld", &n, &m);
  long a, b, w;
```

```
  long k = 0;
  for (long i = 0; i < m; i++){
    fscanf(f,"%ld %ld %ld", &a, &b, &w);
    if (a != b)  e[k++] = Edge(a, b, w);
  }
  m = k;
  fclose(f);
}


int main(){
  init();
  Kruskal();

  long gr = group(1);
  for (long i = 1; i < n; i++){
    if (group(i) != gr){
      rez = 0;
      break;
    }
  }

  FILE *f;
  f = fopen("stabla.out", "w");
  fprintf(f, "%ld\n", rez);
  fclose(f);

  return 0;
}
```

zadatak: Dolina

Planinarski savez južne Srbije je odlučio da sagradi novu atrakciju na Staroj Planini. Po prvi put, svet će videti skijašku dolinu, stazu koja ide i nizbrdo i uzbrdo. Oni veruju da će skijaši dostići dovoljnu brzinu pri spustu, kako bi se popeli u drugom delu staze (pod uslovom da početna visina nije manja od krajnje). Da bi uživanje bilo što veće, potrebno je napraviti najdužu takvu stazu.
Da bi pojednostavili problem, Savez prestavlja teren matricom MxN polja, a visina svakog polja je poznata -- preuzeta iz Geografskog Instituta Srbije. Skijaška dolina je onda niz susednih polja, koja se ne ponavljaju, tako da visine polja prvo samo opadaju duž niza, a potom, počev od nekog polja, visine samo rastu do kraja niza. Dva polja su susedna ako dele zajedničku stranicu. Dužina skijaške doline je broj polja u tom nizu. Matematički preciznije to možemo iskazati na sledeći način. Teren je matrica MxN polja, gde (i, j) predstavlja polje u i-tom redu i j-toj koloni, a $h(i, j)$ je njegova visina. Skijaška dolina je niz (x_1, y_1), (x_2, y_2), ..., (x_l, y_l), takav da:

- za svako i ($1 \le i \le l$-1), važi $x_i = x_{i+1}$ ili $y_i = y_{i+1}$ (susedi),
- ako je $i \ne j$ ($1 \le i, j \le$ l), važi $x_i \ne x_j$ ili $y_i \ne y_j$ (nema ponavljanja), i
- postoji k ($1 \le k \le$ l), tako da važi $h(x_1, y_1) > h(x_2, y_2) > ... > h(x_{k-1}, y_{k-1}) > h(x_k, y_k) < h(x_{k+1}, y_{k+1}) < ... < h(x_l, y_l)$ (dole pa gore).

Dužina ove skijaške doline je l.
Planinarski Savez nema puno iskustva u programiranju, pa te je zamolio da napišeš program koji će naći najdužu skijašku dolinu zadatog terena. Ako postoji više skijaških dolina maksimalne dužine, možeš odabrati bilo koju.
Ulaz:
(Ulazni podaci se nalaze u datoteci **dolina.in**) Prvi red ulaza sadrži cele brojeve M i N, respektivno, razdvojene prazninom ($1 \le M, N \le 70$). U svakom od narednih M redova nalazi se N brojeva, tako da j-ti broj u i-tom redu predstavlja $h(i, j)$. Visine polja su različiti celi brojevi u opsegu $[-10^6, 10^6]$. U svakoj liniji, brojevi su razdvojeni prazninom.
Izlaz:
(Izlazne podatke upisati u datoteku **dolina.out**) U prvi red izlaza potrebno je upisati broj l_{max}, najveću dužinu neke skijaške doline. U sledećih l_{max} linija dajte opis bilo koje skijaške doline te dužine, tako da se u i-tom redu nalaze dva cela broja x_i i y_i razdvojena prazninom, i (x_i, y_i) predstavlja i-to polje doline.

Primer 1:

dolina.in	dolina.out
3 4	9
2 6 7 16	3 1
1 4 3 20	3 2
9 8 17 12	2 2
	2 1
	1 1
	1 2
	1 3
	1 4
	2 4

Primer 2:

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

fajl: dolina.cpp

```
#include <fstream>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;

#define Nmax 70
#define Mmax 70

const int xMove[] = { -1, 1, 0, 0 };
const int yMove[] = { 0, 0, -1, 1 };

int field[Nmax][Mmax];  // visina polja
int sPos[Nmax][Mmax];  // pozicija polja u nizu polja sortiranih po visini
int N, M;

// indeksi polja
struct FieldPointer
{
  int x, y;

  FieldPointer(int xV, int yV)
  {
    x = xV;
    y = yV;
  }
};

bool operator<(FieldPointer fp1, FieldPointer fp2)
{
  return field[fp1.x][fp1.y] < field[fp2.x][fp2.y];
}

// podaci o paru polja:
// - indeksi prvog i drugog polja
// - duzina najduze doline koja se zavrsava ovim poljima
// - prethodni par polja preko koga se doslo do najboljeg resenja
struct FieldPair
{
  __int16 pathLength;
  __int8 move;
```

```
  FieldPair(int plV, int moveV)
  {
    pathLength = plV;
    move = moveV;
  }
};

vector<FieldPointer> sField;  // sortirani niz polja
vector<FieldPair> queuedPair;  // red parova polja u redosledu obilaska

int maxLength, maxPos;
vector<FieldPointer> downHill;  // polja na putu nadole
vector<FieldPointer> upHill;  // polja na putu nagore (u obrnutom redosledu)

void readInput(string fileName)
{
  ifstream inFile(fileName.c_str());
  inFile >> N >> M;
  for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
      inFile >> field[i][j];
  inFile.close();
}

void solve()
{
  int j, p1, p2, x1, y1, x2, y2, mVal, pl, pos1, pos2, pos, xp, yp, move, fieldp;

  // sortiranje polja po rastucoj visini
  for (int k = 0; k < N; k++)
    for (j = 0; j < M; j++)
      sField.push_back(FieldPointer(k, j));
  sort(sField.begin(), sField.end());
  // indeksiranje polja prema poziciji u sortiranom nizu
  for (int k = 0; k < (int)sField.size(); k++)
    sPos[sField[k].x][sField[k].y] = k;

  // *kreiranje reda parova polja*
  // Parovi polja se smestaju u red tako da za dva para polja pp1 i pp2 od kojih se pp1
pojavljuje u redu pre pp2 vazi:
  // - prvo polje u pp1 je na manjoj visini od prvog polja u pp2, ili
  // - drugo polje u pp1 je na majoj visini od drugog polja u pp2.
  // Time se obezbedjuje da, za svaki par polja, svi parovi preko kojih je moglo da se
dodje su se ranije javili u redu
  // i resenje za njih je vec nadjeno.
  for (int k = 0; k < (int)sField.size(); k++)
    for (j = 0; j <= k; j++)
      queuedPair.push_back(FieldPair(-1, -1));

  // obilazak parova polja u redu
  for (int k = 0; k < (int)queuedPair.size(); k++)
  {
    // preuzimanje podataka o paru polja
    p1 = ((int)(sqrt((double)(8 * k + 1)) - 1)) / 2;
    p2 = k - p1 * (p1 + 1) / 2;
    x1 = sField[p1].x;
    y1 = sField[p1].y;
    x2 = sField[p2].x;
    y2 = sField[p2].y;
    mVal = field[x1][y1];

    // specijalni slucaj za par identicnih polja (pocetno polje)
    if ((x1 == x2) && (y1 == y2))
    {
      queuedPair[k].pathLength = 1;
      queuedPair[k].move = -1;
```

```
    }
    else
    {
      // pokusaj pomeranja prvog polja
      for (j = 0; j < 4; j++)
      {
        xp = x1 + xMove[j];
        yp = y1 + yMove[j];
        if ((xp >= 0) && (xp <= N - 1) && (yp >= 0) && (yp <= M - 1))
        {
          fieldp = field[xp][yp];
          if (fieldp < mVal)  // pomeranje je validno samo ako je novo polje na vecoj
visini od oba kraja tekuce doline
                    // kako ne bi doslo do ponavljanja polja
          {
            pos1 = sPos[xp][yp];
            pos2 = sPos[x2][y2];
            if (fieldp < field[x2][y2])
              swap(pos1, pos2);
            pos = (pos1 * (pos1 + 1)) / 2 + pos2;

            pl = queuedPair[pos].pathLength;
            if (pl >= 1)
              if (pl + 1 > queuedPair[k].pathLength)
              {
                queuedPair[k].pathLength = pl + 1;
                queuedPair[k].move = j;
              }
          }
        }
      }
    }
  }

  // nalazenje duzine najduze doline i njenih krajnjih polja
  maxLength = 0;
  for (int k = 0; k < (int)queuedPair.size(); k++)
    if (queuedPair[k].pathLength > maxLength)
    {
      maxLength = queuedPair[k].pathLength;
      maxPos = k;
    }

  // rekonstrukcija doline
  pos = maxPos;
  p1 = ((int)(sqrt((double)(8 * pos + 1)) - 1)) / 2;
  p2 = pos - p1 * (p1 + 1) / 2;
  x1 = sField[p1].x;
  y1 = sField[p1].y;
  x2 = sField[p2].x;
  y2 = sField[p2].y;
  bool switched = false;
  move = queuedPair[pos].move;
  while (queuedPair[pos].move >= 0)
  {
    if (!switched)
      downHill.push_back(FieldPointer(x1, y1));
    else
      upHill.push_back(FieldPointer(x1, y1));
    x1 += xMove[move];
    y1 += yMove[move];
    if (field[x1][y1] < field[x2][y2])
    {
      switched = !switched;
      swap(x1, x2);
      swap(y1, y2);
    }
```

```
    pos1 = sPos[x1][y1];
    pos = (pos1 * (pos1 + 1)) / 2 + sPos[x2][y2];
    move = queuedPair[pos].move;
  }
  downHill.push_back(FieldPointer(x1, y1));  // pocetno polje, moze se smestiti u put
nadole ili obrnuti put nagore
}

void writeOutput(string fileName)
{
  ofstream outFile(fileName.c_str());
  outFile << maxLength << endl;
  int i;
  for (i = 0; i < (int)downHill.size(); i++)
    outFile << downHill[i].x + 1 << " " << downHill[i].y + 1 << endl;
  for (i = (int)upHill.size() - 1; i >= 0; i--)
    outFile << upHill[i].x + 1 << " " << upHill[i].y + 1 << endl;

  outFile.close();
}

int main()
{
  readInput("dolina.in");
  solve();
  writeOutput("dolina.out");
  return 0;
}
```

zadatak: Sanke

Učiteljica Danka je odlučila da svoje đake odvede na sankanje. Međutim, pošto je jedino ona bila raspoložena za takvu avanturu, i đaci ostalih razreda su odlučili da se pridruže. I tako je ona sa N učenika otišla na obližnju planinu. Kada su došli, ona je zamolila svakog đaka da joj kaže odakle će početi da se sanka, i u kom pravcu će se spuštati. Kako je Danka veoma pametna učiteljica, ona je na osnovu reljefa planine i te dve informacije zaključila i koliko brzo će ići sanke svakog učenika. Pošto veoma dugo radi kao učiteljica, ima dovoljno iskustva da sve đake može da kontroliše čak i ako se ne pomera (tj. sve vreme stoji na samo jednom mestu). Danka slabo vidi, a bezbednost dece je na prvom mestu, pa je odlučila da ponese naočare. Ali ne bilo kakve naočare, već specijalne naočare na kojima može da se namesti koliko daleko se vidi s njima. Da bi bila sigurna da je sve u redu, odlučila je da podesi daljinu naočara tako da bar u jednom momentu vidi bar K učenika. Pošto ste Vi njen najbolji učenik, zamolila vas je da joj izračunate kolika je najmanja moguća daljina na koju treba da podesi naočare tako da to važi.

Ulaz:

(Ulazni podaci se nalaze u datoteci **sanke.in**) U prvom redu datoteke se nalaze dva broja x i y ($0 \leq x, y \leq 10^4$) i oni predstavljaju koordinate učiteljice. U drugom redu se nalaze brojevi n ($5 \leq n \leq 1000$) i k ($1 \leq k \leq n$). U svakom od narednih n redova se nalaze 4 broja : prva dva broja $xs[i]$, $ys[i]$ ($0 \leq xs[i], ys[i] \leq 10^4$) označavaju početnu poziciju đaka, a druga dva broja $xk[i]$, $yk[i]$ ($0 \leq xk[i], yk[i] \leq 10^4$) mesto na kome će se đak naći posle 1 sekunde spuštanja.

Napomena: svi đaci se spuštaju pravolinijski, i nijedan đak neće "pregaziti" učiteljicu.

Izlaz:

(Izlazne podatke upisati u datoteku **sanke.out**) U izlaznu datoteku treba ispisati traženu najmanju moguću daljinu na 5 decimala.

Primer 1:

sanke.in	**sanke.out**
6 7 5 4 1 4 1 11 4 12 10 10 4 9 4 7 2 4 10 4 3 1 9 1	5.00000

Primer 2:

sanke.in	**sanke.out**

```
7 7          3.01591
5 3
2 1 2 7
4 1 4 5
6 1 6 10
8 1 8 3
10 1 10 5
```

fajl: sanke.cpp

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

const double Pi = 3.14159265358979323846264338327795;

const long MaxN = 1001;

string fajl_in = "sanke.in";
string fajl_out = "sanke.out";


struct Event{
  double time;
  bool usao;

  Event(double t, bool in):time(t), usao(in){}

  friend bool operator<(const Event &a, const Event &b){
    if (fabs(a.time - b.time) < 1e-9) return a.usao && !b.usao;
    else return a.time < b.time;
  }
};


double cx, cy;

int n;
double mx[MaxN], my[MaxN];
double px[MaxN], py[MaxN];
double speed[MaxN];

int K;


double dist(double x1, double y1, double x2, double y2){
  return sqrt( (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}

bool intersection(double px, double py, double qx, double qy, double r, double &x1, double
&y1, double &x2, double &y2){
  double A = qy - py, B = px - qx, C =  px * qy - py * qx;
  double A1 = -B, B1 = A, C1 = A1 * cx + B1 * cy;

  double d = A * B1 - A1 * B;
  double dx = C * B1 - C1 * B;
  double dy = A * C1 - A1 * C;

  double xpr = dx / d, ypr = dy / d;

  double udaljenost = dist(xpr, ypr, cx, cy);
  if (udaljenost > r + 1e-9) return false;
```

```
  else{
    double alfa = acos(udaljenost / r);

    double tng = atan2(ypr - cy, xpr - cx);
    if (tng < 0) tng += 2 * Pi;

    double u1 = tng - alfa;
    if (u1 < 0) u1 += 2 * Pi;

    double u2 = tng + alfa;
    if (u2 >= 2 * Pi) u2 -= 2 * Pi;

    x1 = cx + r * cos(u1); y1 = cy + r * sin(u1);
    x2 = cx + r * cos(u2); y2 = cy + r * sin(u2);

    return true;
  }
}


bool solve(double r){
  vector<Event> dogadjaj;
  for (int i = 0; i < n; i++){
    double x1, y1, x2, y2;

    if (intersection(mx[i], my[i], px[i], py[i], r, x1, y1, x2, y2)){
      if (dist(mx[i], my[i], cx, cy) < r + 1e-9){   // da li je djak vec unutra
        dogadjaj.push_back(Event(0.0, true));
        if (px[i] != mx[i]){
          if ((x1 - mx[i]) / (px[i] - mx[i]) > 0)  dogadjaj.push_back(Event(dist(x1, y1,
mx[i], my[i]) / speed[i], false));
          else  dogadjaj.push_back(Event(dist(x2, y2, mx[i], my[i]) / speed[i], false));
        }
        else{
          if ((y1 - my[i]) / (py[i] - my[i]) > 0)  dogadjaj.push_back(Event(dist(x1, y1,
mx[i], my[i]) / speed[i], false));
          else  dogadjaj.push_back(Event(dist(x2, y2, mx[i], my[i]) / speed[i], false));
        }
      }
      else if ((px[i] != mx[i] && (x1 - mx[i]) / (px[i] - mx[i]) > 0) || (py[i] != my[i] &&
(y1 - my[i]) / (py[i] - my[i]) > 0)){
        double t1 = dist(x1, y1, mx[i], my[i]) / speed[i];
        double t2 = dist(x2, y2, mx[i], my[i]) / speed[i];
        if (t1 > t2) swap(t1, t2);
        dogadjaj.push_back(Event(t1, true));
        dogadjaj.push_back(Event(t2, false));
      }

    }
  }

  sort(dogadjaj.begin(), dogadjaj.end());

  int unutra = 0;
  for (int i = 0; i < dogadjaj.size(); i++){
    if (dogadjaj[i].usao) unutra++;
    else unutra--;

    if (unutra == K) return true;
  }

  return false;
}


void init(){
  FILE *f;
```

```
  f = fopen(fajl_in.c_str(), "r");
  fscanf(f, "%lf %lf", &cx, &cy);
  fscanf(f, "%d %d", &n, &K);
  for (int i = 0; i < n; i++){
    fscanf(f, "%lf %lf %lf %lf", &mx[i], &my[i], &px[i], &py[i]);
    speed[i] = dist(mx[i], my[i], px[i], py[i]);
  }
  fclose(f);
}

int main(){
  init();

  double l = 0.00001, r = 15000.0;
  for (int korak = 0; korak < 100; korak++){
    double m = (l + r) / 2.0;
    if (solve(m)) r = m;
    else l = m;
  }

  FILE *f;
  f = fopen(fajl_out.c_str(), "w");
  fprintf(f, "%.5lf\n", l);
  fclose(f);

//  int asd; cin >> asd;
  return 0;
}
```

zadatak: Svetiljke

Duž jednog puta koji je prav nalazi se Tomina kuća. Duž tog puta nalazi se i *n* svetiljki, svaka sa

- određenom koordinatom,
- dometom osvetljavanja
- potrošnjom u jedninici vremena.

Tomina kuća ima koordinatu 0. Koordinata svetiljke označava poziciju svetiljke u odnosu na Tominu kuću. Ukoliko je koordinata svetiljke neki broj *x* tada se

- ukoliko je *x* broj veći ili jednak 0 svetiljka nalazi *x* metara desno od Tomine kuće
- ukoliko je *x* manje od 0 svetiljka nalazi *x* metara levo od Tomine kuće

Domet osvetljavanja neke svetiljka je broj koji označava koliko metara levo i desno od te svetiljke je put osvetljen pomoću nje. Ukoliko je domet neke svetiljke neko *x* a njena koordinata neko *y* put je od kordinate *y-x* do *y+x* osvetljen pomoću te svetiljke.
Potrošnja u jedinici vremena neke svetiljke je broj koji označava koliko džula energije u jedinici vremena troši ta svetiljka kada je upaljena.
Na početku su sve svetiljke ugašene. Toma želi, tako što će upaliti neke svetiljke, da osvetli put u kontinuitetu od koordinate *A* do koordinate *B* (*A*<*B*) tako da potrošnja energije u jedinici vremena bude minimalna.
(Toma želi samo da sve tačke puta počev od tačke sa koordinatom *A* do tačke sa koordinatom *B* budu osvetljene, ne zanima ga da li će još neki delovi puta koji su izvan ovog intervala biti osvetljeni)
Ulaz:
(Ulazni podaci se nalaze u datoteci **svetiljke.in**) U prvom redu tekstualne datoteke nalaze se redom brojevi *n*, *A* i *B* (*n* je prirodan broj manji ili jednak 10^5 dok su *A* i *B* celi brojevi koji su veći ili jednaki -$2x10^9$ i manji ili jednaki $2x10^9$ i *A*<*B*) U svakom od sledećih *n* redova nalaze se po tri cela broja koji opisuju jednu svetiljku. Ti brojevi su redom koordinata svetiljke (ceo broj veći ili jednak -10^9 i manji ili jednak 10^9), domet osvetljavanja (prirodan broj manji ili jednak 10^9) i potrošnja u jedinici vremena (prirodan broj manji ili jednak 20000). (U redu čiji je redni broj *i+1* nalazi se opis *i*-te svetiljke)
Izlaz:

(Izlazne podatke upisati u datoteku **svetiljke.out**) U izlaznoj datoteci treba upisati broj koji označava minimalnu potrošnju energije u jedinici vremena koja se mora potrošiti da bi se osvetlio deo puta koji Toma želi ukoliko je to moguće. Ukoliko to nije moguće upisati -1.
U 50% test primera *n* je manje ili jednako 5000.

Primer 1:

svetiljke.in	**svetiljke.out**
6 -10 20 -8 4 2 25 4 7 3 10 1 0 15 4 17 3 5 16 4 2	5

Objašnjenje.
Moguće je osvetliti deo puta od koordinate -10 do koordinate 20. Minimalna potrošnja energije u jedinici vremena koja se mora imati da bi se do izvelo je 5 i ona se dobija paljenjem 1. , 3. i 6. svetiljke. (1. svetiljka osvetljava interval od -12 do -4 , 3. svetiljka osvetljava interval od -7 do 13 a 6. svetiljka osvetljava interval od 12 od 20)

Primer 2:

svetiljke.in	**svetiljke.out**
3 4 20 8 5 10 8 4 7 19 4 10	

fajl: svetiljke.cpp

```
#include <stdio.h>
#include <stdlib.h>

struct el
{
  int levo,desno,potrosnja;
};

el *A;
int n;
int *P,*Min;
int *V;
int levo,desno;
int m;
int br;
el *AA;
int nn;

void unos()
{
  FILE *f;
  f=fopen("svetiljke.in","r");
  fscanf(f,"%d%d%d",&n,&levo,&desno);
  A=(el *)malloc(n*sizeof(el));
  int i,br1,br2;
  for (i=0;i<n;i++)
  {
    fscanf(f,"%d%d%d",&br1,&br2,&A[i].potrosnja);
    A[i].levo=br1-br2;
    A[i].desno=br1+br2;
  }
  fclose(f);
}

void f1()
{
```

```
  int tren=0;
  while (tren<n)
  {
    if (A[tren].desno<=levo || A[tren].levo>=desno)
    {
      A[tren]=A[n-1];
      n--;
    }
    else
    {
      if (A[tren].levo<levo)
        A[tren].levo=levo;
      if (A[tren].desno>desno)
        A[tren].desno=desno;
      tren++;
    }
  }
}

void sort(el *A,int n)
{
  int pom=A[n/2].desno,levo=0,desno=n-1;
  while (levo<=desno)
  {
    while (A[levo].desno<pom)
      levo++;
    while (A[desno].desno>pom)
      desno--;
    if (levo<=desno)
    {
      el pom;
      pom=A[levo];
      A[levo]=A[desno];
      A[desno]=pom;
      levo++;
      desno--;
    }
  }
  if (desno>0)
    sort(A,desno+1);
  if (levo<n-1)
    sort(A+levo,n-levo);
}

void ispis(int broj)
{
  FILE *f=fopen("svetiljke.out","w");
  fprintf(f,"%d",broj);
  fclose(f);
}

int bp(int broj)
{
  int tren,levo=0,desno=m-1;

  while (true)
  {
    tren=(levo+desno)/2;
    if (P[tren]<broj)
      levo=tren+1;
    if (P[tren]>broj)
      desno=tren-1;
    if (P[tren]==broj)
      return tren;
    if (desno<levo)
      if (P[desno]<broj)
        return desno+1;
```

```
      else
        return desno;
  }
}

int mmin(int br1,int br2)
{
  if (br1<br2)
    return br1;
  else
    return br2;
}

void stablo(int m)
{
  br=1;
  while (br<m)
    br*=2;
  V=(int *)malloc((br*2-1)*sizeof(int));
  V[br-1]=0;
  int i;
  for (i=br;i<2*br-1;i++)
    V[i]=2000000000;

  for (i=br-2;i>=0;i--)
    V[i]=mmin(V[i*2+1],V[i*2+2]);
}

int najmanji(int levo,int desno)
{
  levo=br-1+levo;
  desno=br-1+desno;
  int broj=mmin(V[levo],V[desno]);
  int da_levo,da_desno;
  da_levo=levo%2;
  levo=(levo-1)/2;
  da_desno=(desno+1)%2;
  desno=(desno-1)/2;
  while (levo<desno)
  {
    if (da_levo)
      if (V[levo*2+2]<broj)
        broj=V[levo*2+2];
    if (da_desno)
      if (V[desno*2+1]<broj)
        broj=V[desno*2+1];
    da_levo=levo%2;
    levo=(levo-1)/2;
    da_desno=(desno+1)%2;
    desno=(desno-1)/2;
  }
  return broj;
}

void ubaci(int poz,int vred)
{
  V[br-1+poz]=vred;
  int tren=(br-1+poz-1)/2;
  while (tren>0)
  {
    V[tren]=mmin(V[tren*2+1],V[tren*2+2]);
    tren=(tren-1)/2;
  }
  V[0]=mmin(V[1],V[2]);
}

int main()
```

```
{
  unos();
  f1();
  if (n==0)
  {
    ispis(-1);
    return 0;
  }
  if (n>1)
    sort(A,n);
  if (desno>A[n-1].desno)
  {
    ispis(-1);
    return 0;
  }

  int i;
  P=(int *)malloc((n+1)*sizeof(int));
  P[0]=levo;
  P[1]=A[0].desno;
  m=1;
  for (i=1;i<n;i++)
    if (A[i].desno>P[m])
    {
      m++;
      P[m]=A[i].desno;
    }
  m++;
  Min=(int *)malloc(m*sizeof(int));
  Min[0]=0;
  for (i=1;i<m;i++)
    Min[i]=2000000000;
  stablo(m);

  int tren=0;
  for (i=1;i<m;i++)
  {
    bool da=true;
    int broj=0;
    while (da)
    {
      broj++;
      if (tren==n)
        da=false;
      else
        if (A[tren].desno==P[i])
        {
          int br1,br2;
          br1=bp(A[tren].levo);
          br2=bp(A[tren].desno)-1;
          if (br1<=br2)
          {
            int br=najmanji(br1,br2);
            if (br+A[tren].potrosnja<Min[i])
              Min[i]=br+A[tren].potrosnja;
          }
          tren++;
        }
        else
        {
          ubaci(i,Min[i]);
          da=false;
        }
    }
  }
  if (Min[m-1]<2000000000)
    ispis(Min[m-1]);
```

```
  else
    ispis(-1);
  return 0;
}
```

zadatak: Teleport

Usled prevelikog zagađenja vazduha, ljudi su morali da napuste Zemlju i sada žive u svemirskim stanicama. Navedene civilne stanice su numerisane brojevima 1, 2, ..., m. Radi odbrane od vanzemaljaca napravljeno je još *n* vojnih stanica. Između svake vojne i svake civilne stanice moguće je teleportovanje u tačno jednom smeru (ako je moguće teleportovanje iz stanice *A* u stanicu *B* nije moguće iz stanice *B* u *A*). Iz vojne stanice ili nije moguće teleportovanje ni u jednu civilnu stanicu ili je moguće teleportovanje u niz uzastopnih civilnih stanica (npr. iz vojne stanice moguće je teleportovanje u civilne stanice *k, k + 1, ..., p* gde je $1 \leq k \leq p \leq m$). Ne može se teleportovati od jedne civilne stanice do druge civilne, kao i od jedne vojne do druge vojne.
Međutim, vanzemaljci su rešili da osvoje planetu Zemlju. General Đurić je otkrio da vanzemaljci planiraju da napadnu neku stanicu *X*, ali ne zna tačno koju. On je procenio da ako vojsci treba više od 3 teleportovanja do napadnute stanice *X* ona će sigurno pasti u ruke vanzemaljaca. Zato je rešio da nađe sve vojne stanice sa kojih može da stigne da odbrani sve ostale stanice i tamo rasporedi vojsku. Kako je ostalo malo vremena - zamolio je vas mlade programere da mu pomognete.

Ulaz:

(Ulazni podaci se nalaze u datoteci **teleport.in**) U prvoj liniji ulazne datoteke se nalaze dva broja *n* i *m* ($1 \leq n, m \leq 5x10^5$) koji redom predstavljaju broj vojnih i broj civilnih stanica. Zatim sledi *n* linija koje opisuju veze između stanica: u *i*-toj (*i* = 1, 3, ...,*n*) liniji se nalaze dva cela broja *a[i]* i *b[i]* ($0 \leq a[i] \leq b[i] \leq m$) koji predstavljaju krajnje civilne stanice u koje se može teleportovati iz *i*-te vojne stanice (iz vojne stanice moguće je teleportovanje u civilne stanice *a[i], a[i] + 1, ..., b[i]*). Ukoliko je je *a[i]* = *b[i]* = 0 tada se iz date vojne stanice direktno ne može stići ni u jednu civilnu

Izlaz:

(Izlazne podatke upisati u datoteku **teleport.out**) U prvom redu izlazne datoteke se nalazi broj vojnih stanica koje traži general Đurić, a u drugom lista tih vojnih stanica razdvojenih jednim razmakom.

Primer 1:

teleport.in	**teleport.out**
3 4 1 3 2 3 4 4	2 1 3

Objašnjenje.

Vojna stanica 2 ne zadovoljava Đurićevu procenu, jer najkraći putevi do civilne stanice 1 i vojne stanice 1 zahtevaju više od 3 teleportovanja.

Primer 2:

teleport.in	**teleport.out**
4 4 1 4 1 3 0 0 2 4	1 1

fajl: teleport.cpp

```
#include <stdio.h>
#include <stdlib.h>
#define MAXN 500001

int n, m;
int a [MAXN], b [MAXN], p [MAXN];
int count;
int sol [MAXN];

void input ()
{
  FILE *in;
```

```
  in = fopen ("teleport.in", "r");
  fscanf (in, "%d %d", &n, &m);
  for (int i = 0; i < n; i++)
    fscanf (in, "%d %d", &a [i], &b [i]);
  fclose (in);
}

int compare (const void *x, const void *y)
{
  int i = *(int*) x;
  int j = *(int*) y;
  if ((a [i] < a [j]) || ((a [i] == a [j]) && (b [i] > b [j])))
    return -1;
  if ((a [i] > a [j]) || ((a [i] == a [j]) && (b [i] < b [j])))
    return 1;
  return 0;
}

bool covered ()
{
  int right = 0;
  for (int i = 0; i < n; i++)
  {
    if (a [p [i]] > right + 1)
      return false;
    if (right < b [p [i]])
      right = b [p [i]];
  }
  return true;
}

void find_nonested ()
{
  int i = 0;
  while ((i < n) && (a [p [i]] == 0) && (b [p [i]] == 0))
    i++;
  int max = 0;
  while (i < n)
  {
    if (b [p [i]] > max)
    {
      if ((i < n) && !((a [p [i]] == a [p [i + 1]]) && (b [p [i]] == b [p [i + 1]])))
      {
        sol [count] = p [i];
        count++;
      }
      max = b [p [i]];
    }
    i++;
  }
}

void solve ()
{
  for (int i = 0; i < n; i++)
    p [i] = i;
  qsort (p, n, sizeof (int), compare);
  count = 0;
  if (covered () == true)
    find_nonested ();
}

void output ()
{
  FILE *out;
  out = fopen ("teleport.out", "w");
  fprintf (out, "%d\n", count);
```

```
  for (int i = 0; i < count; i++)
    fprintf (out, "%d ", sol [i] + 1);
  fclose (out);
}

int main ()
{
  input ();
  solve ();
  output ();
}
```