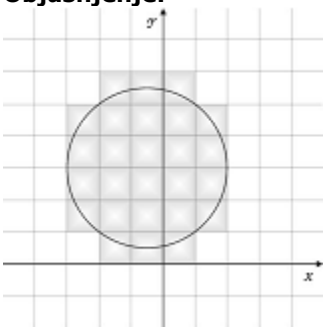
zadatak: Meteor

Farmer Joca je na svojoj beskonačnoj njivi, da bi se lakše snalazio, postavio pravougli koordinatni sistem i povukao prave paralelne koordinatnim osama na međusobnom odstojanju od 1 metar, pri čemu i koordinatne ose pripadaju ovom skupu pravih. Na taj način dobio je polja dimenzije 1x1 metar. I tako je on srećno ubirao plodove, sve dok jednog dana na njegovu njivu nije pao meteor. Na žalost, nije mogao da ga pomeri, pa je odlučio da ne obrađuje polja čiju je unutrašnjost meteorit makar i malo prekrio iz razloga što je on usavršio mašine koja obrađuju cela polja odjednom.
Farmer Joca je detaljnom analizom ustanovio da meteor ima osnovu oblika kruga. Zatim je ustanovio njegov poluprečnik i koordinate centra (mereći u njegovom koordinatnom sistemu). Dajući vam ove podatke, on vas je zamolio da mu pomognete u određivanju površine dela zemljišta koji više neće koristi.

Ulaz:
U prvom i jedinom redu ulaznog tekstualnog fajla **meteor.in** nalaze se tri realna broja *X*, *Y* i *R* sa preciznošću od 2 decimale (-10^6 ≤ *X*, *Y* ≤ 10^6, 0 ≤ *R* ≤ 10^6), razdvojena blanko znakom, koji predstavljaju *X* i *Y* koordinate centra i poluprečnik osnove meteora u metrima.
Izlaz:
U prvi i jedini red izlaznog tekstualnog fajla **meteor.out** upisati ceo broj P koji predstavlja površinu dela zemljišta koju farmer Joca više neće koristiti izraženu u kvadratnim metrima.
Primer:

meteor.in	**meteor.out**
-0.50 3.00 2.50	26

Objašnjenje:



Slika odgovara primeru. Koordinatne ose su obeležene sa *X* i *Y*, a horizontalne i vertikalne linije paralelne njima su na međusobnom odstojanju od 1 metar. Krug odgovara osnovi meteora, a zatamnjena polja odgovaraju onim koja farmer Joca više neće obrađivati.

Rešenje

Zadatak možemo preformulisati na sledeći način. Za zadati krug u ravni je potrebno naći broj celobrojnih kvadratića koji sa krugom imaju presek površine veće od nula. Celobrojni kvadratići su oni omeđeni pravama oblika x=X, x=X+1, y=Y i y=Y+1, gde su X i Y celobrojne konstante. Ovde i nadalje malim slovima označavaćemo koja je koordinata u pitanju, a velikim slovima brojevne vrednosti.
Rešenje zadatka je intuitivno. Potrebno je skanirati krug npr. po z koordinati, posmatrati preseke kruga sa parovima uzastopnih linija skaniranja y=Y i y=Y+1 (gde je Y ceo broj), i na osnovu njih odrediti broj celobrojnih kvadratića između ove dve linije i to dodati ukupnom rezultatu.
Da bismo ovo formalizovali definišimo sledeće pojmove: x-traka(X) je deo ravni omeđen pravama x=X i x=X+1, y-traka(Y) je deo ravni omeđen pravama y=Y i y=Y+1, x-traka(X1) je levo od x-trake(X2) ako je X1 < X2, y-traka(Y1) je niže od y-trake(Y2) ako je Y1 < Y2, analogno definišemo pojmove desno i niže. Nakon ovoga možemo formulisati algoritam za rešavanje zadatka:

```
Y_donja_traka := naći najnižu y-traku koja ima presek sa krugom;
Y_gornja_traka := naći najvišu y-traku koja ima presek sa krugom;
Broj_kvadratića := 0;
for Y := Y_donja_traka to Y_gornja_traka do
begin
  X_leva_traka(Y) := naći najlevlju x-traku koja u y-traci(Y) ima presek sa krugom;
  X_desna_traka(Y) := naći najdesniju x-traku koja u y-traci(Y) ima presek sa krugom;
```

```
    Broj_kvadratića_u_traci := X_desna_traka(Y) - X_leva_traka(Y) + 1;
    Broj_kvadratića := Broj_kvadratića + Broj_kvadratića_u_traci;
  end;
```

U cilju skaniranja kruga po trakama potrebno je naći granične trake, tj. najdonju i najgornju traku koje imaju presek sa njim. Za to nam je potrebno odrediti tačke kruga sa najmanjom i najvećom y koordinatom. Neka su Xc i Yc koordinate centra, a R poluprečnik kruga. "Najniža" tačka kruga ima y koordinatu Y_donja_granica = Yc - R. "Najviša" tačka kruga ima y koordinatu Y_gornja_granica = Yc + R. Slično, "najlevlja" tačka kruga ima x koordinatu X_leva_granica = Xc - R. "Najdesnija" tačka kruga ima x koordinatu X_desna_granica = Xc + R.

Razmotrimo kako na osnovu realnih vrednosti Y_donja_granica i Y_gornja_granica dobiti celobrojne vrednosti Y_donja_traka i Y_gornja_traka. Može se pokazati da je Y_donja_traka najveći ceo broj ne veći od Y_donja_granica, a Y_gornja_traka najveći ceo broj manji od Y_gornja_granica. Razlika je u načinu razrešenja slučaja kada je granica celobrojna. Slično možemo odrediti X_leva_traka kao najveći ceo broj ne veći od X_leva_granica, a X_desna_traka kao najveći ceo broj manji od X_desna_granica.

Ostaje nam da za datu y-traku(Y) nađemo najlevlju i najdesniju x-traku koja ima presek sa krugom. Za to je potrebno naći najlevlju i najdesniju tačku kruga u toj y-traci. S obzirom na geometriju kruga, sve y-trake možemo razvrstati u tri grupe: prvu čini traka koja sadrži centar kruga - centralna traka (ako centar kruga ima celobrojnu y koordinatu centralna traka može biti bilo koja od dve granične, a mogu se i obe posmatrati kao centralne). Drugu grupu čine trake iznad centralne; treću grupu čine trake ispod centralne. Za svaku y-traku(Y) koja je iznad centralne najlevlja i najdesnija tačka kruga dobijaju se u preseku kružnice i prave y=Y, na osnovu kojih (odnosno njihovih x koordinata) se nalaze X_leva_traka(Y) i X_desna_traka(Y). Za svaku y-traku(Y) koja je ispod centralne najlevlja i najdesnija tačka kruga dobijaju se u preseku kružnice i prave y=Y+1. Za centralnu traku najlevlja i najdesnija tačka kruga imaju x koordinate X_leva_granica i X_desna_granica, a najlevlja i najdesnija traka su X_leva_traka i X_desna_traka. Centralnu traku dobijamo na sledeći način: Centralna_traka je najveći ceo broj manji (ne veći) od Yc. Razlika je u tome što kad je Yc ceo broj u slučaju da se upotrebi relacija "manji" bira se donja, a u slučaju relacije "ne veći" bira se gornja od dve ravnopravne trake.

Konačno, rešenje možemo iskazati sledećim algoritmom:

```
  Algoritam_za_rešavanje_zadatka_meteor
  Ulaz: Xc, Yc, R
  Izlaz: Broj_kvadratića
  begin
    Y_donja_granica := Yc - R;
    Y_gornja_granica: = Yc + R;
    Y_donja_traka := najveći_ceo_broj_ne_veći_od( Y_donja_granica);
    Y_gornja_traka := najveći_ceo_broj_manji_od(Y_gornja_granica);
    X_leva_granica := Xc - R;
    X_desna_granica: = Xc + R;
    X_leva_traka := najveći_ceo_broj_ne_veći_od(X_leva_granica);
    X_desna_traka := najveći_ceo_broj_manji_od(X_desnA_granica);
    Centralna_traka := najveći_ceo_broj_manji_od(Yc);
    Broj_kvadratića := X_desna_traka - X_leva_traka + 1; /*inicijalizacija centralnom
trakom*/
    for Y := Y_donja_traka to Centralna_traka - 1 do
    begin
      X_leva_granica(Y) i X_desna_granica(Y) su x koordinate preseka kružnice i prave
y=Y+1;
      /* X_leva_granica(Y) < X_desna_granica(Y) */
      X_leva_traka(Y) := najveći_ceo_broj_ne_veći_od(X_leva_granica(Y));
      X_desna_traka(Y) := najveći_ceo_broj_manji_od(X_desna_granica(Y));
      Broj_kvadratića_u_traci := X_desna_traka(Y) - X_leva_traka(Y) + 1;
      Broj_kvadratića := Broj_kvadratića + Broj_kvadratića_u_traci;
    end;
    for Y := Centralna_traka + 1 to Y_gornja_traka do
    begin
      X_leva_granica(Y) i X_desna_granica(Y) su x koordinate preseka kružnice i prave y=Y;
      /* X_leva_granica(Y) < X_desna_granica(Y) */
      X_leva_traka(Y) := najveći_ceo_broj_ne_veći_od(X_leva_granica(Y));
      X_desna_traka(Y) := najveći_ceo_broj_manji_od(X_desna_granica(Y));
      Broj_kvadratića_u_traci := X_desna_traka(Y) - X_leva_traka(Y) + 1;
      Broj_kvadratića := Broj_kvadratića + Broj_kvadratića_u_traci;
    end;
  end.
```

Složenost: Vremenska složenost rešenja je O(ceo_deo(R)), a memorijska složenost O(const).

Implementacione pojedinosti: Kod realizacije funkcija za određivanje najvećeg celog broja koji je manji od datog realnog broja i najvećeg celog broja koji nije veći od datog realnog broja potrebno je posebno razmatrati pozitivne, a posebno negativne brojeve zbog specifičnosti funkcije odsecanja, a nulu priključiti jednom od ova dva slučaja.
Primetimo da rešenje ne zavisi od celog dela koordinata centra kruga, već samo od razlomljenog dela, jer se translacijom koordinata za ceo broj po nekoj od koordinata ono ne menja. To znači da ove koordinate možemo svesti tako da im je ceo deo nula. Ovo doprinosi manjim vrednostima realnih brojeva sa kojima se radi, što daje i manju grešku izvršenja operacija nad njima. To naravno nije presudno u rešavanju zadatka, ali je interesantno spomenuti.
Broj_kvadratića može prevazilaziti 32-bitni format. Zato je potrebno koristiti 64-bitne brojeve. Umesto toga moguća je i realizacija sa dva 32-bitna broja, od kojih jedan čuva količnik a drugi ostatak pri deljenju rezultata sa nekom velikom 32-bitnom konstantom. Maksimalni rezultat ne prevazilazi (2*ceo_deo(R+1))2 = (2*106)2 = 4*1012, pa se, na primer, može uzeti konstanta 108.
Ulazni podaci su dati sa preciznošću od 8 dekadnih cifara. S obzirom da 8-bajtni brojevi sa pokretnom zapetom imaju preciznost od 15 dekadnih cifara, iz predostrožnosti od grešaka zaokruživanja nakon operacija množenja i kvadriranja, preporučuje se upotreba 10-bajtnog proširenog formata brojeva sa pokretnom zapetom koji ima preciznost od 19 dekadnih cifara.

fajl: meteor.c

```
/*
ZADATAK: meteor
JEZIK: c
*/

#include <stdio.h>
#include <math.h>

#define floattype double
#define breaks 100000000
#define step 8
#define eps 1E-10

FILE *inFile, *outFile;
floattype Cx, Cy, R;
long rez, rezV;

void ucitavanje() {
  fscanf(inFile, "%lf %lf %lf", &Cx, &Cy, &R);
  Cx -= (long)Cx;
  Cy -= (long)Cy;
}

int jednako(floattype a, long b) {
  if (fabs(a - (floattype)b) < eps) return 1;
  else return 0;
}

// nalazi najveci ceo broj manji od datog broja
long findmax(floattype value) {
  long ret;
  if (value > 0)
  {
    ret = (long) value;
    if (jednako(value, ret)) ret--;
  }
  else ret = (long) (value - 1);
  return ret;
}

// nalazi najveci ceo broj ne veci od datog broja
long findmin(floattype value) {
  long ret;
  if (value >= 0) ret = (long) value;
```

```
  else
  {
    ret = (long) (value - 1);
    if (jednako(value, ret + 1)) ret++;
  }
  return ret;
}

void obrada() {
// deklaracije promenljivih
  floattype maxX, maxY, minX, minY, d, Xmin, Xmax;
  long maxIX, maxIY, minIX, minIY, CIy, yI, XminI, XmaxI;

// odredjivanje granica
  maxY = Cy + R;
  minY = Cy - R;
  maxX = Cx + R;
  minX = Cx - R;
  maxIY = findmax(maxY);
  minIY = findmin(minY);
  maxIX = findmax(maxX);
  minIX = findmin(minX);
  CIy = findmax(Cy);

// izracunavanje - glavni deo
  rez = maxIX - minIX + 1;
  rezV = 0;
  for (yI = CIy + 1; yI <= maxIY; yI++)
  {
    d = pow(pow(R, 2) - pow(yI - Cy, 2), 0.5);
    Xmin = Cx - d;
    Xmax = Cx + d;
    XmaxI = findmax(Xmax);
    XminI = findmin(Xmin);
    rez += XmaxI - XminI + 1;
    if (rez >= breaks)
    {
      rezV += rez / breaks;
      rez = rez % breaks;
    }
  }
  for (yI = CIy - 1; yI >= minIY; yI--)
  {
    d = pow(pow(R, 2) - pow(yI + 1 - Cy, 2), 0.5);
    Xmin = Cx - d;
    Xmax = Cx + d;
    XmaxI = findmax(Xmax);
    XminI = findmin(Xmin);
    rez += XmaxI - XminI + 1;
    if (rez >= breaks)
    {
      rezV += rez / breaks;
      rez = rez % breaks;
    }
  }
}

void stampanje() {
  long i, b = breaks;
  if (rezV > 0)
  {
    fprintf(outFile, "%ld", rezV);
    for (i = 0; i < step; i++)
    {
      b /= 10;
      fprintf(outFile, "%ld", rez / b);
      rez = rez % b;
```

```
    }
    fprintf(outFile, "\n");
  }
  else fprintf(outFile, "%ld\n", rez);
}

long main() {
  inFile = stdin;//fopen("meteor.in", "r");
  outFile = stdout;//fopen("meteor.out", "w");

  ucitavanje();
  obrada();
  stampanje();

//    fclose(inFile);
//    fclose(outFile);

  return 0;
}
```

zadatak: Štampa

Mika je vlasnik uspešne izdavačke kuće. Posao mu je cvetao dok jednog dana njegov štampar nije zatvorio štampariju zbog nagomilanih dugova rasipničke porodice. Mika je morao da pronađe drugu štampariju pa se obratio svom školskom drugu Joci. Jocina štamparija, međutim, nije savremena, pa se štampa odvija principom zamena na sledeći način: prvo se postavi početno slovo, pa se primeni određen broj koraka takvih da se u svakom koraku primenjuje neka od metoda zamene, kojom se jedno slovom menja nizom slova. Miki se ovaj način štampe u početku jako svideo, ali je kasnije shvatio da možda ovim načinom ne mogu da se dobiju sve kombinacije parova slova. Pomozite Miki da sazna koje parove susednih slova može da odštampa Jocina štamparija.

Ulaz:
U prvom redu ulaznog tekstualnog fajla **stampa.in** nalaze se dva prirodna broja, broj zamena N i broj slova u jednoj zameni K. U sledećih N redova opisane su zamene na sledeći način: prvo slovo u redu je slovo koje se menja, zatim sledi razmak i posle razmaka se nalaze K slova kojima se menja početno slovo. Sva slova su mala slova engleske abecede i važi $1 < N \leq 10000$, $0 < K \leq 100$. U poslednjem, N+2-om redu nalazi se početno slovo.

Izlaz:
U prvi red izlaznog tekstualnog fajla **stampa.out** upisati X - broj parova susednih slova koja se mogu dobiti na opisani način, a zatim u sledećih X redova upisati parove slova, po jedan par u redu, u leksikografskom poretku.

Primer:

stampa.in	**stampa.out**
5 2	10
a bg	ab
g ab	ba
s dr	bb
a ab	bf
b bf	bg
a	fa
	fb
	ff
	fg
	gb

fajl: stampa.pas

```
{
zadatak: stampa
```

```
jezik: pascal
}
program stampa;
type
  TSusedi=array['a'..'z','a'..'z'] of Boolean;
var
  poc,zav,sus:TSusedi;
  s:array[1..10000] of string;
  n,k,i,t,z,j:integer;
  a,b,slovo:char;
  proveren:array['a'..'z'] of Boolean;
  prva:array['a'..'z'] of integer;
  slova:array[1..30] of char;
procedure sort(l,r:integer);
{quicksort}
var
  i,j:integer;
  k,t:string;
begin
  i:=l;
  j:=r;
  k:=s[(i+j) div 2];
  while i<=j do
  begin
    while s[i]<k do inc(i);
    while k<s[j] do dec(j);
    if i<=j then
    begin
      t:=s[i];
      s[i]:=s[j];
      s[j]:=t;
      inc(i);
      dec(j)
    end
  end;
  if i<r then sort(i,r);
  if l<j then sort(l,j)
end;
procedure PostaviSusede(a,b:char);
{glavna procedura, ispituje koji parovi mogu biti susedni}
var
  j:char;
begin
  if  not Sus[a, b] then
  begin
    Sus[a, b] := true;
    for j := 'a' to 'z' do
    if Zav[a, j] then
      postaviSusede(j, b);
    for j := 'a' to 'z' do
    if Poc[b, j] then
      postaviSusede(a, j);
  end
end;
begin
{inicijalizacija}
  for a:='a' to 'z' do
  begin
    proveren[a]:=false;
    prva[a]:=0;
    for b:='a' to 'z' do
    begin
      poc[a,b]:=false;
      zav[a,b]:=false;
      sus[a,b]:=false
    end
  end;
```

```
{ucitavanje}
{  assign(input,'stampa.in');
  reset(input);}
  readln(n,k);
  for i:=1 to n do
  begin
    readln(s[i]);
    poc[s[i,1],s[i,3]]:=true;
    zav[s[i,1],s[i,k+2]]:=true
  end;
  readln(slovo);
{  close(input);}
{sortiranje i preprocesiranje}
  sort(1,n);
  prva[s[1,1]]:=1;
  for i:=2 to n do
  if s[i,1]<>s[i-1,1] then
    prva[s[i,1]]:=i;
{ispitivanje slova, prvo je pocetno}
  t:=0;
  z:=1;
  slova[z]:=slovo;
  proveren[slovo]:=true;
  repeat
  inc(t);
  slovo:=slova[t];
  i:=prva[slovo];
  if i>0 then
    while s[i,1]=slovo do
    begin
      for j:=1 to k-1 do
      begin
        PostaviSusede(s[i,2+j],s[i,2+j+1]);
        if not proveren[s[i,2+j]] then
        begin
          inc(z);
          slova[z]:=s[i,2+j];
          proveren[s[i,2+j]]:=true
        end
      end;
      if not proveren[s[i,2+k]] then
        begin
          inc(z);
          slova[z]:=s[i,2+k];
          proveren[s[i,2+k]]:=true
        end;
      inc(i)
    end;
  until t=z;
{konacno prebrojavanje i ispis}
{  assign(output,'stampa.out');
  rewrite(output);}
  t:=0;
  for a:='a' to 'z' do
  for b:='a' to 'z' do
    if sus[a,b] then
    inc(t);
  writeln(t);
  for a:='a' to 'z' do
  for b:='a' to 'z' do
    if sus[a,b] then
    writeln(a,b);
{  close(output)}
end.
```

zadatak: Države

Na planeti *X* postoji *N* država i svaka ima po *M* gradova. Dva grada iz neke države mogu biti povezana jednosmernim putem. Povezanost gradova u nekoj državi određena je težinskom matricom. Ovakva matrica je dimenzija *M*M* i vrednost u matrici na mestu sa koordinatama (i, j) je dužina (u kilometrima) puta koji vodi od grada i do grada j. Ako ne postoji put koji vodi od grada i do grada j onda je na mestu (i, j) u matrici vrednost 0. Postoje još i međunarodni putevi koji su jednosmerni sa osobinom, da međunarodni put povezuje neki grad iz zemlje i ($1 \leq i \leq N$-1) sa nekim gradom iz zemlje i+1 i to tako da je smer puta od zemlje i ka zemlji i+1. Između bilo koje dve uzastopne zemlje i ($1 \leq i \leq N$-1) i i+1 postoji tačno *K* međunarodnih puteva. Dužina svakog puta je manja od 10000 kilometara.
Potrebno je startujući iz bilo kog grada na planeti *X* obići sve gradove na zadatoj planeti tako da se svaki grad poseti tačno jednom i da dužina tako napravljene putanje bude minimalna moguća.

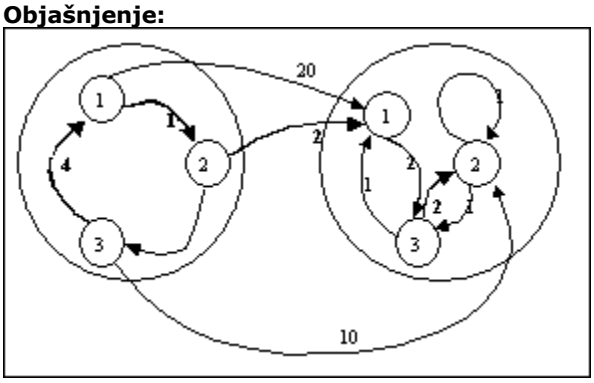
Ulaz:
U prvom redu ulazne datoteke nalaze se tri broja *N*, *M* i *K* međusobno razdvojena razmakom ($2 \leq N \leq 100$, $M \leq 8$, $K \leq M^2$). Zatim je u narednih *M* redova zadata težinska matrica prve zemlje (u svakom redu po jedna vrsta matrice). U narednih *K* redova zadate su međunarodne veze između gradova prve i druge zemlje po formatu grad_prve_zemlje grad_druge_zemlje dužina_veze_u_kilometrima. Zatim je zadata težinska matrica druge zemlje pa međunarodne veze između gradova druge i treće zemlje i tako dalje zaključno sa težinskom matricom *N* te zemlje.

Izlaz:
U prvom i jedinom redu izlazne datoteke treba ispisati dužinu tražene putanje minimalne moguće dužine.

Primer:

drzave.in	**drzave.out**
2 3 3	11
0 1 0	
0 0 1	
4 0 0	
1 1 20	
2 1 2	
3 2 10	
0 0 2	
0 1 1	
1 2 0	

Objašnjenje:



fajl: drzave.c

```
/*
ZADATAK: drzave
JEZIK: c
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MCON 10

typedef enum{false,true} Bool;
```

```
long a[101][MCON][MCON], b[101][MCON][MCON], act[MCON];
long N,M,K,Rez,BrPer;
short GenPer[40500][MCON];
const char *ulaz= "drzave.in";
const char *izlaz="drzave.out";



void Citaj();
void Init();
void Uradi();
void GPer(int);
void GGPer();
void Ispisi();

int main(){

  Init();
  Citaj();
  GGPer();
  Uradi();
  Ispisi();
  return 0;
}

void Ispisi(){
  FILE *fout=stdout;//fopen(izlaz,"w");
  fprintf(fout,"%ld",Rez);
//    fclose(fout);
}
void Uradi(){
  int i,j,l;
  long min;

  for(i=N;i>0;i--){
    GPer(i);  //nakon GPer u act se nalaze minimalne duzine putanja od
          //bilo kog grada drzave i.

    for(j=1;j<=M;j++){a[i-1][M+1][j]=0; a[i-1][j][M+1]=0;}

    for(j=1;j<=M;j++)
      for(l=1;l<=M;l++)
        if(b[i-1][j][l]>0 && act[l]>0) {
          if (a[i-1][j][M+1]==0){a[i-1][j][M+1]=b[i-1][j][l]+act[l];}
          else {
            if (a[i-1][j][M+1]>b[i-1][j][l]+act[l])
              a[i-1][j][M+1]=b[i-1][j][l]+act[l];
          }
        }
    if (i!=1) for(j=1;j<=M+1;j++) act[j]=0;
  }
  min=0;
  for(i=1;i<=M;i++)
    if((act[i]>0) &&((min>act[i])||(min==0))) min=act[i];
  Rez=min;
}

void GPer(int broj){
  long per[MCON];
  Bool ok;
  int i,j;
  long zbir,br=0;

  for(i=0;i<BrPer;i++){
    for(j=1;j<=M+1;j++) per[j]=GenPer[i][j];
    //provera da li je putanja moguca
    zbir=0; ok=true;
```

```
    for(j=2;j<=M+1;j++)
      if ((a[broj][per[j-1]][per[j]]>0)||(a[broj][per[j-1]][per[j]]==0 && j==M+1 &&
broj==N) )
        zbir+=a[broj][per[j-1]][per[j]];
      else ok=false;
    if ((ok==true)&&(((act[per[1]]>zbir)&&(act[per[1]]!=0))||(act[per[1]]==0))) {
      act[per[1]]=zbir;
    }
    //kraj provere
  }
}
void GGPer(){
  short per[MCON];
  Bool bio[MCON], ok;
  int i,j,k,l;

  for(i=1;i<=M+1;i++) {per[i]=i; bio[i]=true;}
  BrPer=0;

  for(;;){
    for(i=1;i<=M+1;i++) GenPer[BrPer][i]=per[i];
    BrPer++;
    //sledeca permutacija
    k=M; ok=false;
    while((k>0)&&(ok==false)){
      for(i=per[k]+1;i<=M;i++)
        if(bio[i]==false) {
          bio[per[k]]=false;
          per[k]=i;
          bio[i]=true;
          ok=true;
          for(j=k+1;j<=M;j++)
            for(l=1;l<=M;l++)
              if(bio[l]==false){
                bio[l]=true; per[j]=l;
                break;
              }
        }
        if (ok==false){
          bio[per[k]]=false;
          per[k]=0; k--;
        }
    }
    if (k==0) break;
  }
}

void Init(){
  int i,j,l;

  for(i=1;i<=M;i++) {
    act[i]=0; a[N][i][M+1]=0; a[N][M+1][i]=0;
  }
  for(i=1;i<=N;i++)
    for(j=1;j<=M;j++)
      for(l=1;l<=M;l++){
        a[i][j][l]=0;
        if (i<N) b[i][j][l]=0;
      }
}

void Citaj(){
  FILE *fin=stdin;//fopen(ulaz,"r");
  int g1,g2,d,i,j,l;

  fscanf(fin,"%d%d%d",&N,&M,&K);
  for(i=1;i<=N;i++){
```

```
    for(j=1;j<=M;j++)
      for(l=1;l<=M;l++) fscanf(fin,"%d",&a[i][j][l]);
    if(i!=N){
      for(j=0;j<K;j++){
        fscanf(fin,"%d%d%d",&g1,&g2,&d);
        b[i][g1][g2]=d;
      }
    }
  }
//    fclose(fin);
}
```

zadatak: Jezerca

Perica već nekoliko meseci provodi dane kraj računara igrajući igru Jezerca. Na početku igre se na ekranu pojavi pravougaona mreža kvadratića na kojoj je prikazan reljef nekog predela u obliku stubića raznih visina koji se nižu jedan za drugim. Zatim preko cele širine ekrana počinje padati kiša. Ona popunjava udubljenja u reljefu, a višak nestaje na levom i desnom kraju ekrana (tj. na stubićima na levom i desnom kraju se uopšte ne zadržava). Kiša prestaje kada se popune sva udubljenja (tako da sva kiša koja padne odlazi na krajeve ekrana i gubi se). Voda koja je napunila udubljenja obrazuje određen broj jezera (vodene površine odvojene bar jednim vertikalnim stubom od drugih vodenih površina).
Perica u tom trenutku treba da izračuna koliko ima kvadratića popunjenih vodom u onom jezeru u kome se zadržalo najviše vode (tj. ima najviše popunjenih kvadratića). Napiši program koji će Perici pomoći da odredi broj kvadratića u jezeru sa najviše kvadratića.

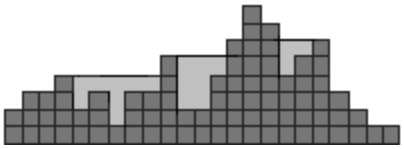
Ulaz:
U prvom redu ulaznog fajla **jezerca.in** nalazi se ceo broj n ($1 \leq n \leq 100000$) i to je ukupan broj stubića u reljefu. U svakom od sledećih n redova nalazi se po jedan ceo broj i oni predstavljaju visine stubića u redosledu u kome se oni prikazani na ekranu (od levog kraja prema desnom kraju). Visine stubića su između 1 i 10000.

Izlaz:
U prvi red izlaznog fajla **jezerca.out** treba ispisati broj kvadratića popunjenih vodom u jezeru u kome ima najviše kvadratića popunjenih vodom. Ako ne postoji niti jedno jezero, štampati broj nula (0).

Primer:

jezerca.in	jezerca.out
23 2 3 3 4 2 3 1 3 3 5 2 2 4 6 8 7 4 5 6 3 2 1 1	8

Objašnjenje:



Slika prikazuje izgled reljefa, kao i raspored jezera nakon popunjavanja svih udubljenja. Kao što se sa slike vidi ima tri jezera koja imaju 8, 7 i 3 popunjena kvadratića.

fajl: jezerca.c

```
/*
ZADATAK: jezerca
JEZIK: c
*/
# include <stdio.h>

# define MAXN 1000000

long n;
long mz;
int a[MAXN], ml[MAXN], md[MAXN];

int citaj() {
  long i;
  scanf("%ld", &n);
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
  return(0);
}

void resi() {
  long k;
  long i;
  long mm;
  long tz;
  for (ml[0] = 0, i = 1; i < n; i++)
    if (a[i-1] > ml[i-1]) ml[i] = a[i-1]; else ml[i] = ml[i-1];
  for (md[n-1] = 0, i = n-2; i >= 0; i--)
    if (a[i+1] > md[i+1]) md[i] = a[i+1]; else md[i] = md[i+1];
  mz = 0; tz = 0;
  for (k = 0, i = 1; i < n-1; i++) {
    if (ml[i] < md[i]) mm = ml[i]; else mm = md[i];
    if (mm <= a[i]) {
      if (tz > mz) mz = tz;
      tz = 0;
    } else {
      tz += mm-a[i];
    }
  }
  if (tz > mz) mz = tz;
}

void pisi() {
  printf("%ld\n", mz);
}

int main() {
  if (citaj()) {
    exit(1);
  }
  resi();
  pisi();
  return 0;
}
```

zadatak: Mars

Pošto je istraživanje Marsa sve popularnije, nacionalna svemirska agencija SCG Space Agency je pre određenog vremena poslala sopstvene robote, kojih ima *n*, na Mars. Ovi roboti se kreću po površini Marsa, skupljaju uzorke kamenja i slično, i šalju podatke nazad na Zemlju. Međutim, pošto koriste zastareli hardver, neki od robota su se već pokvarili. SCG Space Agency ne može direktno da sazna koji su roboti u kvaru, jer oni i dalje šalju nazad podatke. Jedini način da se to sazna je da se nekom robotu pošalje zahtev da izvrši inspekciju nekog drugog robota. Ali, problem je što robot koji vrši inspekciju takođe može da bude u kvaru - u tom slučaju rezultat inspekcije nije validan. Preciznije, ako robot A vrši inspekciju robota B, onda je rezultat inspekcije dat u tabeli.

A	B	Rezultat inspekcije
ispravan	ispravan	B je ispravan
ispravan	neispravan	B je neispravan
neispravan	ispravan	rezultat je slučajan
neispravan	neispravan	rezultat je slučajan

Pošto je urađeno *m* različitih inspekcija, SCG Space Agency želi da sazna spisak svih robota za koje se sigurno može tvrditi da su neispravni, kako bi pokrenula proceduru njihovog samouništenja.

Ulaz:

U prvom redu ulaznog fajla **mars.in** nalaze se brojevi *n* i *m* ($n \leq 1000$, $m \leq 10^5$). U svakom od sledećih *m* redova nalaze se podaci o jednoj inspekciji. To su redom brojevi A, B i R. A je indeks robota koji vrši inspekciju, dok je B indeks robota nad kojim se vrši inspekcija. Roboti su numerisani brojevima od 1 do *n*. R je rezultat inspekcije - 0 označava da je robot B ispravan, a 1 da je neispravan.

Izlaz:

U prvi red izlaznog fajla **mars.out** treba ispisati broj robota za koje se sa sigurnošću tvrdi da su neispravni. U drugom redu treba da se nalazi spisak tih robota, sortiran po indeksu robota.

Primer:

mars.in	mars.out
5 6 1 2 0 2 3 0 3 1 1 5 1 0 1 4 1 5 4 0	2 1 5

Objašnjenje:

Ako pretpostavimo da je robot 1 ispravan, sledi da su i roboti 2 i 3 ispravni, na osnovu prve i druge inspekcije. Ali zbog treće inspekcije imamo kontradikciju, pa sledi da je robot 1 sigurno neispravan. Zbog toga i četvrte inspekcije robot 5 je takođe sigurno neispravan. O ostalim robotima ne možemo ništa da zaključimo.

fajl: Mars.java

```
import java.io.*;
import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

public class Mars {

    static BufferedInputStream in;
    static StreamTokenizer tok;
    static PrintStream out;

    static int nextInt() throws Exception {
        tok.nextToken();
        return (int) tok.nval;
    }

    public static final int RUNNING = 0;
    public static final int BROKEN = 1;
```

```
static class Edge {
    int v;
    int result;
    public Edge(int v, int result) {
        this.v = v;
        this.result = result;
    }
}

int n;
List[] edges;
boolean[][] conn0;
boolean[][] conn1;

void readInput() throws Exception {
    n = nextInt();

    edges = new List[n];
    for (int i = 0; i < n; i++) {
        edges[i] = new ArrayList();
    }
    conn0 = new boolean[n][n];
    conn1 = new boolean[n][n];

    int m = nextInt();
    for (int i = 0; i < m; i++) {
        int u = nextInt() - 1;
        int v = nextInt() - 1;
        int result = nextInt();

        if (result == 0) {
            if (conn0[u][v])
                throw new Exception();
            conn0[u][v] = true;
            edges[u].add(new Edge(v, result));
        } else if (!conn1[u][v]) {
            conn1[u][v] = true;
            conn1[v][u] = true;
            edges[u].add(new Edge(v, result));
            edges[v].add(new Edge(u, result));
        }
    }
}

boolean[] broken;

boolean[] visited;
int[] mark;
int[] queue;

boolean bfs(int source) {
    Arrays.fill(visited, false);
    int head = 0, tail = 0;

    queue[0] = source;
    mark[source] = RUNNING;
    visited[source] = true;

    while (head <= tail) {
        int u = queue[head++];

        for (Edge e : (List<Edge>) edges[u]) {
            if (broken[e.v] && e.result == RUNNING)
                return true;

            if (visited[e.v]) {
```

```
                    if (mark[e.v] == RUNNING && e.result == BROKEN)
                        return true;
                    if (mark[e.v] == BROKEN && e.result == RUNNING)
                        return true;
                } else {
                    mark[e.v] = e.result;
                    if (e.result == RUNNING) {
                        visited[e.v] = true;
                        queue[++tail] = e.v;
                    }
                }
            }
        }
        return false;
    }

    List brokenList = new ArrayList();

    void solve() {
        broken = new boolean[n];

        visited = new boolean[n];
        mark = new int[n];
        queue = new int[n];

        for (int source = 0; source < n; source++)
            if (!broken[source])
                if (bfs(source)) {
                    broken[source] = true;
                    brokenList.add(source);
                }

        Collections.sort(brokenList);
    }

    void printOutput() {
        out.println(brokenList.size());
        for (int i : (List<Integer>) brokenList)
            out.print((i+1) + " ");
        out.println();
    }

    public static int DEFAULT_TEST_INDEX = 1;

    public static void main(String[] args) throws Exception {
//        int testIndex = args.length > 0 ? Integer.parseInt(args[0]) : DEFAULT_TEST_INDEX;
//        String inpath = String.format("mars.%02d.in", testIndex);
//        String outpath = String.format("mars.%02d.1.out", testIndex);
//
//        in = new BufferedInputStream(new FileInputStream(inpath));
//        out = new PrintStream(new BufferedOutputStream(new FileOutputStream(outpath)));
//        tok = new StreamTokenizer(in);

        in = new BufferedInputStream(System.in);
        out = System.out;

        Mars m = new Mars();
        m.readInput();
        m.solve();
        m.printOutput();

//        in.close();
//        out.close();
    }
}
```

zadatak: Sef

Zadaci za savezno takmičenje iz informatike čuvaju se u sefu visoke sigurnosti. Sef može da ima više šifara za otvaranje, od kojih je svaka niz sastavljen od malih slova engleskog alfabeta. Kako organizatori takmičenja nemaju poverenja jedni u druge, svako od njih je dobio da čuva samo delimičnu informaciju o šiframa, i šifre se mogu rekonstruisati tek kada se svi organizatori skupe zajedno.
Svaki organizator je dobio šemu koja je niz sastavljen od slova ("a"..."z"), upitnika ("?") i zvezdica ("*"). Pojava nekog slova u šemi odgovara pojavi istog tog slova u šiframa. Pojava upitnika u šemi zamenjuje proizvoljno slovo u šiframa, a pojava zvezdice u šemi zamenjuje proizvoljan broj slova (uključujući i nula) u šiframa. Šeme date organizatorima su takve da se slažu sa svakom od šifara
Neposredno pred početak takmičenja organizatori su se okupili da otvore sef, međutim pošto su previše uzbuđeni zbog početka takmičenja, to im nikako ne polazi za rukom. Zato su zamolili smirene takmičare da im pomognu da pronađu neku od šifara sa najmanjim mogućim brojem slova, kako bi takmičenje počelo sa što manjim zakašnjenjem.

Ulaz:

Dato vam je 10 ulaznih fajlova sa imenima sef.*nn*.in gde je *nn* broj test primera (01, 02, ..., 10) .
U prvom redu ulaznog fajla zapisan je broj *m* - broj organizatora. U svakom od narednih *m* redova zapisana je najpre dužina šeme *m*-tog organizatora, potom tačno jedan razmak, i na kraju šema zapisana bez razmaka među znakovima.

Izlaz:

Za dati ulazni fajl **sef.*nn*.in** napraviti izlazni fajl u čiji prvi red treba zapisati "`# sef, nn`" (bez navodnika) gde umesto *nn* treba zapisati broj test primera. U drugi red upisati dužinu najkraće šifre, a u treći bilo koju najkraću šifru.

Primer:

ulaz (sef.00.in)	**izlaz**

`3`
`4 ?a*b`
`4 a*b*`
`5 *a?a*`	