Zadaci sa rešenjima Mala olimpijada 2006.

zadatak: Spajder Đura

Spajder Đura se nekako našao na Novom Beogradu. Nakon što je ponovo spasao svet od uništenja, rešio je da ruča. Spajder Đura, naravno najviše voli ćevape, i to u velikim količinama. Novi Beograd je, kao što je svima poznato, podeljen na blokove (*N* x *N*), i u svakom bloku se nalazi po jedna ćevabdžinica. Spajder Đuri su poznate cene ćevapa u svakom od blokova. Međutim, kako je Spajder Đura umoran, on može da preleti (kačeći se o zgrade, pomoću paukove mreže) najviše *K* blokova. Iz svakog bloka, on može da skoči (preleti) na bilo koji od osam susednih (znači može da se kreće dijagonalno).
Spajder Đura je čuo da se u Beogradu održava Mala olimpijada pa je rešio da takmičare priupita za pomoć, ne bi li pojeo najviše moguće ćevapa, to jest našao najjeftiniju ćevabdžinicu u okolini u kojoj se nalazi. Na nesreću, Spajder Đura ne zna gde se trenutno nalazi, pa takmičari moraju da za svaki blok pronađu najjeftiniju ćevabdžinicu udaljenu najviše *K* blokova.
Vaš zadatak je da za unetu matricu, koja predstavlja cene ćevapa u svakom od blokova, odredite matricu istih dimenzija čiji elementi predstavljaju najnižu cenu ćevapa u svim blokovima do kojih se može stići sa najviše *K* skokova (letenja).
Ulaz:
U tekstualnom faju **spider.in** se u prvom redu nalaze brojevi *N* i *K*, ($1 \le N \le 2000$, $1 \le K \le N$). Zatim se u narednih *N* redova nalazi matrica veličine *N* x *N*, i to u svakom redu po *N* celih brojeva iz intervala [0..2000000000].
Izlaz:
U tekstualni fajl **spider.out** u *N* redova ispisati matricu dimenzija *N* x *N* (koja predstavlja rešenje goreopisanog problema), i to u svakom od *N* redova po *N* brojeva razmaknutih blanko znakom.
Primer:

spider.in	spider.out
4 1 1 2 3 4 6 5 4 3 2 3 4 5 8 7 6 5	1 1 2 3 1 1 2 3 2 2 3 3 2 2 3 4

fajl: spider.cpp

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

#define MAX_N     2000
#define MAX_PRICE 2000000000L

using namespace std;

int Prefix[MAX_N*2], Suffix[MAX_N*2];

vector<int>& LinearMin(vector<int> &A, int k) {

  vector<int> &Sol = *(new vector<int>(A.size()));

  for (int i=0;i<A.size();i++)
    Suffix[i] = (i%(2*k+1)==0)?A[i]:min(Suffix[i-1], A[i]);

  Prefix[A.size()-1] = A[A.size()-1];
  for (int i=A.size()-2;i>=0;i--)
    Prefix[i] = ((i+1)%(2*k+1)==0)?A[i]:min(Prefix[i+1], A[i]);

  for (int i=0;i<A.size();i++)
    Sol[i] = min((i-k>=0)?Prefix[i-k]:Suffix[i],(i+k<A.size())?Suffix[i+k]:Prefix[i]);

  return Sol;
}
```

```
int main() {
  FILE *fin, *fout;

  fin = fopen("spider.in", "r");
  fout = fopen("spider.out", "w");

  int n, k;
  fscanf(fin, "%d %d", &n, &k);
  vector<int> A(n+k), Sol[MAX_N];

  for (int j=0;j<n;j++) {

    for (int i=0;i<n;i++)
      fscanf(fin, "%d", &A[i]);
    for (int i=0;i<k;i++)
      A[i+n] = MAX_PRICE;

    vector<int> LineSol = LinearMin(A, k);
    for (int i=0;i<n;i++)
      Sol[i].push_back(LineSol[i]);
  }

  for (int i=0;i<n;i++) {
    for (int j=0;j<k;j++)
      Sol[i].push_back(MAX_PRICE);
    Sol[i] = LinearMin(Sol[i], k);
  }

  for (int j=0;j<n;j++) {
    for (int i=0;i<n;i++)
      fprintf(fout, "%d ", Sol[i][j]);
    fprintf(fout, "\n");
  }
  fclose(fin);
  fclose(fout);
}
```

fajl: spider_rbtree.cpp

```
#include <stdio.h>
#include <vector>
#include <map>

#define MAX_N     2000
#define MAX_PRICE 2000000000L

using namespace std;

vector<int>& RBTreeMin(vector<int> &A, int k) {

  vector<int> &Sol = *(new vector<int>(A.size()));

  map<int, int> M;

  for (int i=0;i<A.size()+k;i++) {
    if (i<A.size())
      M[A[i]]++;
    if (i > 2*k) {
      M[A[i-2*k-1]]--;
      if (M[A[i-2*k-1]] == 0)
  M.erase(A[i-2*k-1]);
    }
    if (i >= k)
      Sol[i-k] = M.begin()->first;
  }
```

```
  return Sol;
}


int main() {

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

  int n, k;
  fscanf(fin, "%d %d", &n, &k);
  vector<int> A(n), Sol[MAX_N];

  for (int j=0;j<n;j++) {

    for (int i=0;i<n;i++)
      fscanf(fin, "%d", &A[i]);

    vector<int> LineSol = RBTreeMin(A, k);
    for (int i=0;i<n;i++)
      Sol[i].push_back(LineSol[i]);
  }

  for (int i=0;i<n;i++)
    Sol[i] = RBTreeMin(Sol[i], k);

  for (int j=0;j<n;j++) {
    for (int i=0;i<n;i++)
      fprintf(fout, "%d ", Sol[i][j]);
    fprintf(fout, "\n");
  }
  fclose(fin);
  fclose(fout);
}
```

fajl: spider_basic.cpp

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

#define MAX_N     2000
#define MAX_PRICE 2000000000L

using namespace std;

int Mat[MAX_N][MAX_N], SolX[MAX_N][MAX_N], SolY[MAX_N][MAX_N];

int main() {

  int n, k;
  scanf("%d %d", &n, &k);

  for (int j=0;j<n;j++)
    for (int i=0;i<n;i++)
      scanf("%d", &Mat[j][i]);

  for (int j=0;j<n;j++)
    for (int i=0;i<n;i++) {
      SolX[j][i] = Mat[j][i];
      for (int l=max(0,i-k);l<min(n,i+k+1);l++)
  SolX[j][i] <?= Mat[j][l];
    }
```

```
  for (int j=0;j<n;j++)
    for (int i=0;i<n;i++) {
      SolY[i][j] = SolX[i][j];
      for (int l=max(0,i-k);l<min(n,i+k+1);l++)
  SolY[i][j] <?= SolX[l][j];
    }


  for (int j=0;j<n;j++) {
    for (int i=0;i<n;i++)
      printf("%d ", SolY[j][i]);
    printf("\n");
  }
}
```

zadatak: Loto

U zemlji Srećiji zavladala je nova pošast – nagradna igra na sreću Loto. Pravila ove igre su sledeća: iz bubnja koji sadrži n kuglica numerisanih brojevima od 0 do n-1 sve kuglice se izvlače nekim redosledom i pobednik je onaj koji pogodi tačan redosled kuglica. Bubanj je oblika horizontalno postavljene kružne ploče na kojoj se nalaze kuglice Iznos nagrade koja se dodeljuje pobedniku računa se kao zbir brojeva oblika $a_i*10000^{n-i}$, gde a_i označava broj kuglice koja je izašla i-ta po redu.
Prošlo je već mnogo vremena od kada je Loto startovao a još se nije pojavio nijedan srećni dobitnik, i to je nagnalo ljude da počnu da sumnjaju u verodostojnost igre. Međutim, dok se ostali samo žale kako je sve to nameštено, profesor Đurić, stari vuk igara na sreću, je izračunao da je iznos eventualnog dobitka veći nego što bi ga koštala uplata svih mogućih kombinacija, pa je odlučio da na taj način zaradi. Nestrpljivo iščekujući dan kada će konačno biti dodeljena premija, možete zamisliti naše zaprepašćenje kada su u naše prostorije banuli predstavnici organizatora ove igre i rekli nam da je sve zaista nameštено (1:0 za narod), i da nas mole, pošto na sledećem izvlačenju svakako moraju da dodele premiju (1:0 za profesora Đurića), da im pomognemo da iznos nagrade bude što manji.
Loto se namešta na sledeći način: ispod bubnja, paralelno sa njegovom ravni, nalazi se jak magnet u obliku šipke, koji se može postaviti u proizvoljan položaj. U tačno određenom momentu magnet se aktivira što uzrokuje da se sve kuglice momentalno prilepe za njega (kuglica na magnet doleće pod pravim uglom) i u tom redosledu izađu iz bubnja.
Naše zaprepašćenje još uvek traje, a pošto nismo sigurni ni koliko je sve to moralno, odlučili smo da problem predamo vama – odredite kombinaciju koja će organizatore koštati što manje.

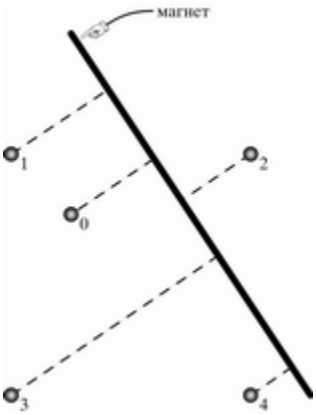
Ulaz:
U prvoj liniji ulaznog fajla **loto.in** se nalazi prirodan broj n ($1 \le n \le 3000$), broj kuglica. U narednih n linija se nalaze celi brojevi x_i, y_i razdvojeni prazninom ($-15000 \le x_i, y_i \le 15000$), koordinate kuglice i u momentu uključivanja magneta. Kuglice su opisane redom, počev od kuglice sa oznakom 0, do kuglice sa oznakom n-1.

Izlaz:
U n linija izlaznog fajla **loto.out** upisati brojeve kuglica u redosledu u kom izlaze pri najpovoljnijem položaju magneta, tako da se u i-tom redu nalazi broj kuglice koja izlazi i-ta po redu.

Primer:

loto.in	loto.out
5 1 3 0 4 4 4 0 0 4 0	1 0 2 3 4

Objašnjenje



Minimalnu vrednost glavnog dobitka postići ćemo ako magnet postavimo kao na slici. Glavni dobitak u tom slučaju iznosi: **1**·10000^4 + **0**·10000^3 + **2**·10000^2 + **3**·10000 + **4**

fajl: loto.pas

```
const
  fin = 'loto.in';
  fout = 'loto.out';
  MaxN = 3000;

type
  TVector = record
    x, y : Longint;
    id : Integer;
  end;

var
  n : Integer;
  p : array[0..MaxN] of TVector;
  min : array[1..MaxN] of Integer;



   function Compare(const a, b : TVector) : Longint;
   begin
     if a.x <> b.x then
       Compare := a.x - b.x
     else
       Compare := a.y - b.y;
   end;


   function Prod(const a, b : TVector) : Shortint;
   var
      p : Longint;
   begin
     p := a.x*b.y - a.y*b.x;

      if p < 0 then
         Prod := -1
      else
         if p > 0 then
            Prod := 1
         else
            if (a.x*b.x > 0) or (a.y*b.y > 0) then
               Prod := 0
            else
               Prod := 2;
   end;
```

```
 function Sub(const a, b : TVector) : TVector;
 var
   c : TVector;
 begin
   c.x := a.x - b.x;
   c.y := a.y - b.y;
   Sub := c;
 end;


 function Perp(const a : TVector) : TVector;
 var
   c : TVector;
 begin
   c.x := -a.y;
   c.y := a.x;
   Perp := c;
 end;



 var
    m, t : TVector;

 procedure Sort(l, r : Integer);
 var
   i, j : Integer;
 begin
   i := l;
   j := r;
   m := p[(l + r) div 2];
   repeat
     while Compare(p[i], m) < 0 do Inc(i);
     while Compare(p[j], m) > 0 do Dec(j);
     if i <= j then
     begin
       t := p[i];
       p[i] := p[j];
       p[j] := t;
       inc(i);
       dec(j);
     end;
   until i > j;

   if l < j then Sort(l, j);
   if i < r then Sort(i, r);
 end;


procedure Solve;
var
  k, i, j : Integer;
  m, mc : Integer;
  c : array[0..MaxN+1] of Integer;
  cn, d : Integer;
    boundL, boundH : TVector;
  t1, t2 : TVector;
    valid : Boolean;
begin
  Sort(1, n);

  for k := n downto 2 do
  begin
       // nalazenje konveksnog omotaca (tacke su vec sortirane)
    cn := 1;
```

```
    c[1] := 1;
    d := 1;
    for i := 2 to 2*k-1 do
    begin
      j := k - Abs(k - i);
      while (cn > d) and (Prod(Sub(p[c[cn]], p[c[cn-1]]), Sub(p[j], p[c[cn]])) <= 0) do
        Dec(cn);
      Inc(cn);
      c[cn] := j;
      if i = k then d := cn;
    end;
    dec(cn);
       c[0] := c[cn];


    // nalazenje dozvoljene tacke na konveksnom omotacu sa najmanjom oznakom
       m := 0;
    p[0].id := n + 1;
    for i := 1 to cn do
    begin
          if (k = n) then
             valid := true
          else
          begin
             t1 := Perp(Sub(p[c[i]], p[c[i-1]]));
           t2 := Perp(Sub(p[c[i+1]], p[c[i]]));

             if Prod(boundL, t1) < 0 then
                valid := Prod(boundL, t2) = 1
             else
                valid := Prod(t1, boundH) > 0;
          end;

          if valid and (p[c[i]].id < p[m].id) then
      begin
        m := c[i];
        mc := i;
      end;
       end;

       min[n-k+1] := p[m].id;


       // suzavamo dozvoljeni interval pravaca
       t1 := Perp(Sub(p[c[mc]], p[c[mc-1]]));
    t2 := Perp(Sub(p[c[mc+1]], p[c[mc]]));

       if (k = n) or (Prod(t2, boundH) > 0) then boundH := t2;
    if (k = n) or (Prod(t1, boundL) < 0) then boundL := t1;


    // izbacivanje tacke m iz daljeg razmatranja (tacke ostaju sortirane)
       for i := m to k-1 do
          p[i] := p[i+1];
    end;
    min[n] := p[1].id;
end;


procedure ReadInput;
var
  f : Text;
  i : Integer;
begin
  Assign(f, fin);
  Reset(f);
  Readln(f, n);
```

```
    for i := 1 to n do
    begin
      Readln(f, p[i].x, p[i].y);
      p[i].id := i - 1;
    end;
    Close(f);
  end;


  procedure WriteOutput;
  var
    f : Text;
    i : Integer;
  begin
    Assign(f, fout);
    Rewrite(f);
    for i := 1 to n do
      Writeln(f, min[i]);
    Close(f);
  end;



begin
  ReadInput;
  Solve;
  WriteOutput;
end.
```

zadatak: Roštilj

Jedan grad na samom jugu Bajtovije je nadaleko poznat po izvanrednom roštilju. Jednom godišnje, održava se festival, kada svi najbolji majstori roštilja dođu i takmiče se u tome ko će bolje pripremiti neke od poslastica: pljeskavice, vešalice, ćevape, ražnjiće, kobasice, batake, pečenje... Svake godine u to vreme vlada nezapamćeni haos u gradu, gužve na ulicama, zakrčenje saobraćaja.
Draganče je predsednik organizacionog komiteta Roštiljijade. On želi da organizuje dva kamiona punih mesa, koji bi se kretali gradom i prodavali đakonije. Zamišljeno je da kamioni krenu zajedno sa jedne raskrsnice i nađu se na drugoj, gde bi se prebrojalo ko je prodao više. Da kamioni ne bi bili jedan drugome direktna konkurencija, odlučeno je da ne smeju proći kroz istu raskrsnicu. Draganče ima mapu grada. Pomozite mu da pronađe dve raskrsnice koje omogućavaju ovakvu akciju.
Poznat je broj raskrsnica, *N*, ne veći od 5000, kao i to da broj ulica nije veći od 200000. Svaka ulica spaja dve raskrsnice. Neke ulice su jednosmerne, a neke dvosmerne. Sve raskrsnice su numerisane brojevima od 1 do *N*.

Ulaz:
U prvoj liniji ulaznog fajla **rostilj.in** se nalazi broj *N*. Zatim sledi *N* linija. U *J*-toj (*J* = 2, 3, …, *N*+1) liniji se nalazi najpre prirodan broj *K*, koji označava da za raskrsnicu *J*-1 postoji *K* raskrsnica do kojih se direktno (jednom ulicom) može stići iz raskrsnice *J*-1. Potom sledi *K* brojeva iz intervala 1..*N*, razdvojenih razmakom, koji označavaju te raskrsnice.

Izlaz:
U prvoj i jedinoj liniji izlaznog fajla **rostilj.out** treba ispisati dva broja *X* i *Y* koji kazuju da postoje dva disjunktna puta od *X* do *Y*. Putevi su disjunktni ako ne postoji raskrsnica (osim prve i poslednje) koja se nalazi i na jednom i na drugom putu. Garantuje se da će takve dve raskrsnice uvek postojati. U slučaju da postoji više rešenja, ispisati bilo koje. Red završiti oznakom za kraj reda.

Primer:

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

Objašnjenje:
Jedan način je 1 -> 2 -> 4, a drugi 1 -> 3 -> 4.

Rešenje

Zadatak se može rešiti na više načina. Najpre ćemo prezentovati rešenje komisije, a potom i ostala rešenja koja su se pokazala jednako dobrim, ako ne i boljim u datom slučaju. Rešenja će biti predstavljena korišćenjem odgovarajuće terminologije iz teorije grafova, kojom se pretpostavlja da učenici više ili manje barataju. Ukoliko ima nejasnoća ili nepoznatih termina, preporučuju se knjige iz oblasti algoritama, kao i Internet kao sveobuhvatna literatura.
Rešenje komisije: Najpre ćemo podeliti graf na jako povezane komponente što se može jednostavno učiniti obilaskom grafa u dubinu i numeracijom čvorova. Problem delimo na dva dela, jedan je da tražimo rešenje kada su oba čvora u istoj kopmponenti, a drugo kada su u različitim komponentama.

1. Tražimo rešenje u okviru jedne komponente. Izaberemo proizvoljan čvor u komponenti i napravimo dfs stablo od čvorova iz date komponente, sa korenom u izabranom čvoru. Pošto je komponenta jako povezana, dakle, postoji put iz korena do svih ostalih čvorova, svi čvorovi komponente će biti u dfs stablu. Svaka ivica unapred ("forward") ili poprečna ivica ("cross") koju pronađemo u komponenti nam sa sigurnošću daje rešenje. Njih ćemo detektovati i razlikovati na osnovu dfs brojeva koje dodeljujemo čvorovima, kao i markera čvorova koji nam govore da li je određeni čvor obiđen ili nije. Što se tiče povratnih ("back") ivica, sa njima treba vršiti posebnu proveru, zato što jedna povratna ivica ne donosi rešenje. Rešenje ćemo pronaći ako postoji čvor tako da postoje dve povratne ivice koje počinju u njegovim potomcima. To ćemo činiti tako što za svaki čvor pamtimo da li postoji povratna ivica koja počinje u njegovim potomcima i tu informaciju prosleđivati roditelju tog čvora. Kada pronađemo dve povratne ivice, pronašli smo i rešenje, koje rekonstruišemo uz pomoć informacije o roditeljima.
2. Tražimo rešenje između komponenata. Konstruišemo novi graf koji smo dobili od početnog tako što je svaka komponenta zaseban čvor, a ivica između dve komponente postoji ukoliko postoji ivica između jednog čvora prve i jednog čvora druge komponte. Jasno je da novodobijeni graf neće imati ciklusa, jer bi postojanje ciklusa značilo da komponente nisu dobijene na pravi način. S obzirom da je novi graf acikličan, na njemu možemo primeniti algoritam topološkog sorta. Krenuvši od čvorova koji imaju izlazni stepen 0, za svaki čvor pamtimo listu čvorova čiji je on predak i tu informaciju prosleđujemo njegovim roditeljima. Kada se desi da je neki čvor roditelj dva različita čvora koji su preci jednom istom čvoru, rešenje je pronađeno. Ono što još preostaje je da se pronađu realni čvorovi u datim komponentama koji su početak i završetak ovih puteva. Oni se pronalaze tako što se čvorovi koji su nosioci ivica među komponentama pamte još u startu formiranja grafa komponenata. Još bi trebalo voditi računa u slučajevima kada iz jedne komponente u drugu postoje dve ivice, što nam odmah donosi rešenje.

Ukupna složenost ovog algoritma je u najgorem slučaju $O(n^2 + m)$, gde je n broj čvorova, a m broj grana u grafu.
Zadatak je uspešno, sa maksimalnih 100 poena rešilo četiri takmičara, Ilić Andreja, Rajko Nenadov, Stevan Jončić i Slobodan Mitrović. Svaka čast! Ovaj zadatak poseduje i daleko jednostavnije rešenje (rešenja) od zvaničnog rešenja komisije i ta rešenja su upravo implementirali ovi takmičari. Pa ipak postoji razlog zbog kog je gore navedeno rešenje zvanično: Teoretska složenost ovih algoritama u najgorem slučaju je $O(n*m)$. Međutim, u praksi se pokazuje da je jako teško ili gotovo nemoguće napraviti test primere koji razlikuju ova dva rešenja.
Rešenja takmičara su se zasnivala na bfs i dfs pretragama i formiranju odgovarajućih stabala koja su se formirala sa korenom u svakom čvoru. Na sličan način kao u zvaničnom rešenju kada se traži rešenje u jednoj komponenti (prvi slučaj), traže se poprečne ivice i ivice unapred i tako pronalazi rešenje. Svakako, težina zadatka i jeste u tome da se primeti činjenica da ako graf poseduje mnogo ivica, rešenje će brzo biti pronađeno, a ako poseduje malo ivica, onda je n približno sa m, čime se i smanjuje vreme izvršavanja programa.

fajl: rostilj.cpp

```
#include <stdio.h>
#include <vector>
#include <stack>
#include <queue>
#include <algorithm>

using namespace std;
const int MAXN = 5010;
```

```
vector<int> v[MAXN], kvb[MAXN];
vector<pair<int,int> > tc[MAXN];
int outdg[MAXN];
int n, sccn, dfsn;
int white = 0, grey = 1, black = 2;
int mark[MAXN];
int order[MAXN], low[MAXN], scc[MAXN], prev[MAXN];
int b[MAXN][MAXN];  // grane
int t[MAXN][MAXN];


stack<int> s;
FILE* f;

void init(){
     freopen("rostilj.in","r", stdin);
     f = fopen("rostilj.out","w");
}

void read(){
  int k;
    scanf("%d",&n);
    for (int i = 0; i < n; i++){
        scanf("%d",&k);
        outdg[i] = k;
        for (int j = 0; j < k; j++){
            int a;
            scanf("%d", &a);
            v[i].push_back(a-1);
        }
    }
}

void dfs_strongly(int k){
  s.push(k);
  low[k] = order[k] = dfsn++;
  mark[k] = grey;
  for (int i = 0; i < v[k].size(); i++)
    if (mark[v[k][i]] == white){
      dfs_strongly(v[k][i]);
      low[k] = min(low[k], low[v[k][i]]);
    }
    else{
      if (scc[v[k][i]]==-1) low[k] = min(low[k], order[v[k][i]]);
    }
  if (low[k] == order[k]){
    int x;
    do{
      x = s.top();
      s.pop();
      scc[x] = sccn;
    } while(x != k);
    sccn++;
  }
}

void finish(int a, int b){
  fprintf(f, "%d %d\n", a+1, b+1);
  fclose(stdin);
  fclose(f);
  exit(0);
}

void divide_components(){
  memset(mark, 0, sizeof(mark));
  memset(scc, -1, sizeof(scc));
```

```
  sccn = 0; dfsn = 0;
  for (int i = 0; i < n; i++)
    if (!mark[i]){
      dfs_strongly(i);
    }
  memset(outdg, 0, sizeof(outdg));
  memset(b, 0, sizeof(b));
  for(int i = 0; i < n; i++)
    for (int j = 0; j < v[i].size(); j++)
      if (!b[scc[i]][scc[v[i][j]]] && scc[i]!=scc[v[i][j]]){
        b[scc[i]][scc[v[i][j]]] = 1;
        kvb[scc[v[i][j]]].push_back(scc[i]);
        outdg[scc[i]]++;
      }
      else if (scc[i]!=scc[v[i][j]]) finish(i,v[i][j]);
}

int common_ancestor(int a, int b, int* prev){
  int m1[MAXN];
  memset(m1,0,sizeof(m1));
  while(1){
    if (a>-1){
      if (m1[a])
        return a;
      m1[a] = 1;
      a = prev[a];
    }
    if (b>-1){
      if (m1[b])
        return b;
      m1[b] = 1;
      b = prev[b];
    }
  }
}


int dfs_inside(int k){
  order[k] = dfsn++;
  mark[k] = grey;
  int up = -1;
  for (int i = 0; i < v[k].size(); i++)
    if (scc[v[k][i]] == scc[k])
      if (mark[v[k][i]] == white){
        prev[v[k][i]] = k;
        int up1 = dfs_inside(v[k][i]);
        if (up1 >= 0)
          if (order[up1]<order[k])
            if (up < 0)
              up = up1;
            else
              if (order[up]>order[up1])
                finish(k, up);
              else
                finish(k,up1);

      } else if (mark[v[k][i]]==grey){
        if (up < 0)
          up = v[k][i];
        else
          if (order[up]>order[v[k][i]])
            finish(k,up);
          else
            finish(k,v[k][i]);
      } else{
        finish(common_ancestor(k,v[k][i], prev),v[k][i]);
      }
```

```
  mark[k] = black;
  return up;
}

void find_in_components(){
  memset(mark, 0, sizeof(mark));
  memset(order, 0, sizeof(order));
  memset(prev, -1, sizeof(prev));
  dfsn = 1;
  for (int i = 0; i < n; i++)
    if (!mark[i])
      dfs_inside(i);
}

void finish_components(int a, int b, pair<int,int> d){
  int x, y, i, found = 0;
  for (x=0; x < n && !found; x++){
    if (scc[x] == a){
      for (i = 0; i < v[x].size() && scc[v[x][i]]!=b; i++);
          if (i < v[x].size()) found = 1;
        }
    }
    found = 0;
  for (y=0; y < n && !found; y++){
    if (scc[y] == d.second){
      for (i = 0; i < v[y].size() && scc[v[y][i]]!=d.first; i++);
        if (i < v[y].size()) found = 1;
        }
    }
  y = v[y-1][i];
  finish(x-1,y);
}

void find_among_components(){
  int i;
  queue<int> q;
  int order[sccn], topsortnum = 0;
  memset(t,0,sizeof(t));
  for (i = 0; i < sccn; i++)
    if (outdg[i] == 0)
      q.push(i);
  while(q.size()>0){
    int x = q.front();
    q.pop();
    order[x] = topsortnum++;
    pair<int,int> dest(-1,-1); int source = -1;
    for (i = 0; i < kvb[x].size(); i++){
      int y = kvb[x][i];
      pair<int,int> p(x,y);
      tc[x].push_back(p);
      if (--outdg[y] == 0)
        q.push(y);
      for (int j = 0; j < tc[x].size(); j++){
        if (t[y][tc[x][j].first] == 1)
          if (dest.first == -1)
            {dest = tc[x][j];source=y;}
          else
            {dest = order[dest.first]>order[tc[x][j].first]?dest:tc[x][j];source=y;}
        t[y][tc[x][j].first] = 1;
        tc[y].push_back(tc[x][j]);
      }
      tc[x].pop_back();
    }
    if (dest.first != -1) finish_components(source,x,dest);
  }
}
```

```
int main(){
   init();
     read();
   divide_components();
   find_in_components();
   find_among_components();
   for (int i = 0; i < n ;i++)
     printf("%d\n",scc[i]);
   return 0;
}
```