```
//Aho Corasik
struct Node{
  Node * H[70];Node *sub;Node *subd;
  int pos;
  Node(){
   for(int i = 0  ;i < 70 ; i++)
      H[i] = 0;
   sub = 0;subd = 0;pos = -1;
  }
}Tree;
int NUMBER(char c){}
int B[1000];
void Add(char * C , int p){
  Node * v = &Tree;
  int vv ;
  for(int i = 0 ; C[i] ; i++){
           vv =  NUMBER(C[i]);
           if(!v->H[vv])
               v->H[vv] = new Node();
           v = v->H[vv];
    }
  if(v->pos == -1)
    v->pos = p;
  B[p] = v->pos;
}
void Build(){
  queue<Node *> cola;
  Node * r = &Tree;
  for(int i = 0 ; i < 70 ; i++)
     if(r->H[i]){
        cola.push(r->H[i]);
        r->H[i]->sub = r;
     }else
        r->H[i] = r;
     while (!cola.empty()){
       Node *u = cola.front();
       cola.pop();
       for (int c = 0; c < 70; c++)
         if (u->H[c]){
          cola.push(u->H[c]);
          Node *v = u->sub;
          while (!v->H[c]) v = v->sub;
            u->H[c]->sub = v->H[c];
          if (u->H[c]->sub->pos != -1)
              u->H[c]->subd = u->H[c]->sub;
          else
           u->H[c]->subd = u->H[c]->sub->subd;
          }
      }
}
bool I[10000];int n;
void Search(char *C){
 Node * u = &Tree;
 for (int j = 0; C[j]; j++){
   int c = NUMBER(C[j]);
   while(!u->H[c]) u = u->sub;
   u = u->H[c];
   if (u->pos != -1)
    I[u->pos] = 1;
   for (Node *v = u->subd; v; v = v->subd)
     I[v->pos] = 1;
 }
 for(int i = 0 ; i < n;  i++)
   if(I[B[i]])
      printf("Y\n");
   else
```

```
      printf("N\n");
}
char M[100005],C[2005];
int main(){
     scanf("%s",M);
     Add(M,-1);
     scanf("%d",&n);
     for(int i = 0 ; i < n; i++){
        scanf("%s",C);
        Add(C,i);
     }
    Build();
    Search(M);
    scanf("\n");
  return 0;
}
//Automata
import java.util.*;
public class SuffixAutomaton {
static class State {
     int length;
     int link;
     int endpos;
     Map<Character, Integer> next = new
HashMap<Character, Integer>();
     List<Integer> ilink = new
ArrayList<Integer>();
};
State[] st;
int size;
int last;
int lastp;
void saExtend(char c) {
```

```
 int nlast = size++;
 st[nlast] = new State();
 st[nlast].length = st[last].length + 1;
 st[nlast].endpos = st[last].length;
 int p;
 for (p = last; p != -1 && !
st[p].next.containsKey(c); p = st[p].link)
     st[p].next.put(c, nlast);
 if (p == -1)
  st[nlast].link = 0;
 else {
  int q = st[p].next.get(c);
  if (st[p].length + 1 == st[q].length)
       st[nlast].link = q;
  else {
       int clone = size++;
       st[clone] = new State();
       st[clone].length = st[p].length + 1;
       st[clone].next.putAll(st[q].next);
       st[clone].link = st[q].link;
       for (; p != -1 &&
st[p].next.containsKey(c)
             && st[p].next.get(c) == q; p =
st[p].link)
          st[p].next.put(c, clone);
                    st[q].link = clone;
                    st[nlast].link = clone;
                    st[clone].endpos = -1;
               }
          }
          last = nlast;
     }
public void buildSA(String s) {
     int n = s.length();
     st = new State[Math.max(2, 2 * n - 1)];
     st[0] = new State();
     st[0].link = -1;
     st[0].endpos = -1;
     last = 0;
     size = 1;
     for (char x : s.toCharArray()) {
          saExtend(x);
     }
     for (int i = 1; i < size; i++) {
          st[st[i].link].ilink.add(i);
     }
}
public String lcs(String a, String b) {
     buildSA(a);
     int p = 0;
     lastp = 0;
     int len = 0;
     int best = 0;

     int bestpos = -1;
     for (int i = 0; i < b.length(); ++i) {
          char cur = b.charAt(i);
          if (!st[p].next.containsKey(cur)) {
               for (; p != -1 && !
st[p].next.containsKey(cur); p = st[p].link) {
               }
               if (p == -1) {
                    p = 0;
                    len = 0;
                    continue;
               }
```

```
               len = st[p].length;
          }
          ++len;
          p = st[p].next.get(cur);
          if (best < len) {
               best = len;
               bestpos = i;
               lastp = p;
          }
     }
     return b.substring(bestpos-best+1, bestpos
+ 1);
}
public int[] occurrences(String needle, String
haystack) {
     String common = lcs(haystack, needle);
     if (!common.equals(needle))
          return new int[0];
     List<Integer> list = new
ArrayList<Integer>();
     dfs(lastp, needle.length(), list);
     int[] res = new int[list.size()];
     for (int i = 0; i < res.length; i++)
          res[i] = list.get(i);
     Arrays.sort(res);
     return res;
}
void dfs(int p, int len, List<Integer> list) {
     if (st[p].endpos != -1 || p == 0)
          list.add(st[p].endpos - len + 1);
     for (int x : st[p].ilink)
          dfs(x, len, list);
}
```

```
     public static void main(String[] args) {
          SuffixAutomaton sa = new
SuffixAutomaton();
          System.out.println(sa.lcs("aab1ccc",
"zb1cz"));
          int[] res = sa.occurrences("ab",
"xabaabxabababaxbab");

     System.out.println(Arrays.toString(res));
     }
}
//Tabla pref
void KMP() {int j = F[0] = -1;
  for (int i = 1; i <= strlen(C); ++i){
    while (j>=0 && C[j]!=C[i-1]) j=F[j];
      F[i] = ++j;
  }}
void Radio(){
for(i=0,j=0; i<2*n;i+=k,j=max(j-k,0)){
while(i-j>=0 && i+j+1<2*n && C[(i-
j)/2]==C[(i+j+1)/2])++j;
 R[i]=j;
for(k = 1;i>=k&&R[i]>=k&&R[i-k]!=R[i]-k;++k)
   R[i+k] = min(R[i-k],R[i]-k);
  }}
void compute_pref() {
  PREF[g = 0] = N;
  for (i=1; i<N; ++i) {
    if (i < g && PREF[i - f] != (g - i))
      PREF[i]=min(PREF[i-f], g-i);
    else{
      g = max(g, f = i);
    while (g < N && C[g] == C[g — f])++g;
      PREF[i] = g - f;
   } }}
//Vertex Cover
#define MAX 3000
int g1,g2,m,x,y;
vector<int> Adj[MAX];
int Dist[MAX],Par[MAX];
bool BFS(){
  queue<int>Cola;
  for(int i =1 ; i <=g1+g2 ;++i)
    if(Par[i] == 0){
      Cola.push(i);Dist[i] = 0;
  } else
    Dist[i] = MAX;
  Dist[0] = MAX;
  while(!Cola.empty()){
    int v = Cola.front();Cola.pop();
    for(int i=0 ; i<Adj[v].size(); i++)
      if(Dist[Par[Adj[v][i]]] == MAX){
        Cola.push(Par[Adj[v][i]]);
        Dist[Par[Adj[v][i]]] = Dist[v]+1;
      }
  }
  return Dist[0] !=  MAX;
}
bool DFS(int x){
  for(int i = 0 ;i < Adj[x].size();i++){
    if(Dist[Par[Adj[x][i]]]==Dist[x]+1 &&
(Par[Adj[x][i]]==0 || DFS(Par[Adj[x][i]]))){
      Par[x] = Adj[x][i];
      Par[Adj[x][i]] = x;
      return 1;
    }
  }
  Dist[x] = MAX;
  return 0;
}
set<int> conj;
int n,N[300];
int parL[MAX],parR[MAX],T[MAX];
void VertexCover(){
  memset(parL, -1, sizeof(parL));
  memset(parR, -1, sizeof(parR));
  for(int i = 1 ; i <= g1 ; i++)
    if(Par[i]){
      parL[i] = Par[i];
      parR[Par[i]] = i;
    }
  memset(T, 0, sizeof(T));
  vector<int> L, R;
  for(int i = 1 ; i <= g1 ; i++)
    if(parL[i]==-1){
      L.push_back(i);
      T[i] = 1;
    }
while(L.size()){
    for(int i = 0 ; i<L.size();i++){
      int v = L[i];
      for(int j=0 ; j<Adj[v].size();j++){
        int viz = Adj[v][j];
        if(T[viz]==0 && parL[v]!=iz){
          T[viz] = 1;
          R.push_back(viz);
        }
      }
    }
    L.clear();
    for(int i=0; i<R.size(); i++){
      int v = R[i];
      if(parR[v]>=0 && T[parR[v]]==0){
        T[parR[v]] = 1;
        L.push_back(parR[v]);
      }
    }
    R.clear();
  }
  for(int i=1; i<=g1 ; i++)
    if(T[i] == 0)
      printf("r%d ",i);
  for(int i=1; i<=g2; i++)
    if(T[i+g1]==1)
      printf("c%d ",i);
}
int main(){
  while(true){
    scanf("%d%d%d",&g1,&g2,&n);
    if(g1==0 && g2==0 && n==0)
      break;
    int a , b;
    for(int i = 0 ; i <= g1 + g2 ; i++){
      Adj[i].clear();
      Par[i] = 0;
    }
    for(int i = 0; i < n ; i++){
      scanf("%d%d",&a,&b);
      Adj[a].push_back(g1+b);
    }
    int res = 0;
    while(BFS())
```

```
      for(int i = 1 ; i <=g1+g2+1 ; i++)
        if(!Par[i] && DFS(i))
          res++;
    printf("%d ",res);
    VertexCover();
    printf("\n");
  }
  return 0;
}
```

Número Ciclomático:

M : cantidad de Aristas
N: # de vértices
P:# de componentes conexas.
NC = M – N + P   cantidad de ciclos.

Número de Estabilidad Interna:

Un conjunto de vértices se dice que es interiormente estable si dos vértices cualesquiera del conjunto no son adyacentes. El mayor subconjunto interiormente estable de un grafo es conocido como número de estabilidad interna. Lo designaremos por I.

En todo grafo se cumple la siguiente relación:

I(G) * NC(G) = Total de vértices de la red.

```
//LIS2
map< int, int > M[ MaxN ];
void insert( int m, par a ){
     while( 1 ){
map< int, int > ::
iteratorit=M[m].lower_bound( a.x );
 if( it == M[m].begin() ){
        if( !M[m].size() ) break;
if( it->y > a.y ) { M[m].erase(it);continue; }
break;}if( it == M[m].end() ){
          if( a.y >= ( --it )->y ) return;
       if( a.y < it->y && a.x <= it->x )
          { M[m].erase( it ); continue; }
              break;
          }
int y2 = it->y, y1 = ( --it )->y;
if( a.y >= y1 && a.y >= y2 ) return;
if( a.y < y1 && a.y > y2 ) break;
M[m].erase( ++it );
     }
     M[m].insert( a );
}
int main( void ){int n, ret = 0; bool ok;
    scanf( "%d", &n );
    for( int i = 0; i < n; ++i ){
         par a;
         scanf( "%d %d", &a.x, &a.y );
         int lo = 0, hi = ret;
         for( ; lo < hi; ){
              int mid = ( lo + hi + 1 ) / 2;
map<int,int>::iterator
it=M[mid].lower_bound( a.x );
 if( it == M[mid].begin() )   ok = 0;
 else  ok = ( --it )->x < a.x && it->y < a.y;
 ok ? lo = mid : hi = mid - 1;
      }
 insert( lo + 1, a );ret = max( ret, lo + 1 );
    }
    printf( "%d\n", ret );
    return 0;
}
```

```
//Geometria 2D
struct L: public vector <P>{ //Linea
     L (const P & a, const P & b){
          push_back(a);
          push_back(b);
     }
};
bool operator <(const P &a, const P &b) {
     return real(a)!=real(b)?real(a)<real(b)
     :imag(a)<imag(b);
}
double cross(P a, P b){
     return imag(conj(a)*b);
}
double dot(P a, P b){
     return real(conj(a)*b);
}
int ccw(P a, P b, P c){ //Orient de 3 puntos
     b-=a; c-=a;
     if (cross(b,c) > 0) return +1;  //CC
     if (cross(b,c) < 0) return -1;  //C
     if (dot(b,c) < 0) return +2;    //cab
     if (norm(b)<norm(c)) return -2; //abc
     return 0;
}
bool intersectLL (L l, L m){
return abs(cross(l[1]-l[0], m[1]-m[0])) > EPS
|| abs(cross(l[1]-l[0], m[0]-l[0])) < EPS;
}
bool intersectLS (L l, L s){
return cross(l[1]-l[0], s[0]-l[0])*
cross(l[1]-l[0], s[1]-l[0]) <EPS;
}
```

```
bool intersectLP (L l, P p){
     return abs(cross(l[1]-p, l[0]-p))<EPS;
}
bool intersectSS (L s, L t){
if(abs(s[0]-t[0])<EPS || abs(s[0]-t[1])<EPS ||
abs(s[1]-t[0]) < EPS  || abs(s[1]-t[1]) < EPS)
     return 1;
return ccw(s[0],s[1],t[0])*ccw(s[0],s[1],t[1])
<=0 && ccw(t[0],t[1],s[0])*ccw(t[0],t[1],s[1])
<=0;
}
bool intersectSP (L s,P p){
return abs(s[0]-p)+abs(s[1]-p)-abs(s[1]-s[0])
< EPS;
}
P projection(L l,P p){
     double t = dot(p-l[0],l[0]-l[1]) /
norm(l[0]-l[1]);
     return l[0] + t*(l[0]-l[1]);
}
P reflection(L l, P p){
     return p + (P(2,0) * (projection(l,p)-p));
}
double distanceLP(L l,P p){
     return abs(p - projection(l,p));
}
double distanceLL(L l, L m){
return intersectLL(l,m)?0:distanceLP(l,m[0]);
}
double distanceLS(L l, L s){
     if (intersectLS(l,s)) return 0;
     return min(distanceLP(l,s[0]),
distanceLP(l,s[1]));
```

```
}
double distanceSP(L s, P p){
     const P r = projection(s,p);
     if (intersectSP(s,r)) return abs(r-p);
     return min( abs(s[0]-p), abs(s[1]-p) );
}
double distanceSS (L s, L t) {
  if (intersectSS(s, t)) return 0;
  return min( min( distanceSP(s,t[0]),
distanceSP(s,t[1]) ), min( distanceSP(t,s[0]),
distanceSP(t,s[1])));
}
P crosspoint(L l, L m){
     double A = cross( l[1]-l[0], m[1]-m[0]);
     double B = cross( l[1]-l[0], l[1]-m[0]);
     if (abs(A)<EPS && abs(B)<EPS) return m[0];
     if (abs(A) < EPS) return P(0,0);
     return m[0] + B / A * (m [1] - m [0]);
}
P circunferenceCenter(P a, P b, P c){
     P x = 1.0/conj(b-a), y = 1.0/conj(c-a);
     return (y-x)/(conj(x)*y - x*conj(y))+a;
}
typedef vector<P> Pol;
Pol convexHull(Pol ps){
     int n = ps.size(), k=0;
     sort(ps.begin(), ps.end());
     Pol ch (2*n);
for (int i = 0; i <n; ch [k++] = ps [i++])
while(k>=2&&ccw(ch[k-2],ch[k-1],ps[i])<=0)
--k;
for(int i=n-2,t=k+1;i>=0;ch[k++]=ps[i--])
while(k>=t&&ccw(ch[k-2],ch[k-1], ps[i])<=0)
--k;
     ch.resize(k-1);
     return ch;
}
enum{OUT, ON, IN};
#define next(P,i) P[(i+1)%P.size()]
int pointInPolygon(const Pol &pol, const P &p)
{
     bool in = false;
     for (int i=0; i<pol.size(); i++){
          P a=pol[i]-p, b=next(pol,i)-p;
          if (imag(a)>imag(b)) swap(a,b);
          if (imag(a)<=0 && 0<imag(b))
               if (cross(a,b)<0) in = !in;
          if (cross(a,b)==0 && dot(a,b)<=0)
               return ON;
     }
     return in?IN:OUT;
}
double area(const polygon & P) {
  double A = 0;
  for (int i=0 ; i<P.size(); ++i)
    A += cross(P[i], next(P, i));
  return A;
}
double anguloEjeX(P a){
     P b = P(1,0);
if (dot(b,a)/(abs(a)*abs(b))==1) return 0;
if (dot(b,a)/(abs(a)*abs(b))==-1) return PI;
double aux = asin(cross(b,a)/(abs(a)*abs(b)));
if (a.real() < 0 && a.imag() > 0) aux += PI/2;
if (a.real() < 0 && a.imag() < 0) aux -= PI/2;
if (aux <0) aux += 2*PI;
```

```
     return aux;
}
double anguloEntreVectores(P a, P b){
     double aa = anguloEjeX(a);
     double bb = anguloEjeX(b);
     double r = bb - aa;
     if (r<0) r+=2*PI;
     return r;
}
double anguloEntre3Puntos(P a,P b,P c){ //abc
     a-=b; c-=b;
     return anguloEntreVectores(a,b);
}
pair <P,P> closestPair (vector <P> p) {
     int n = p.size(), s=0, t=1, m=2, S[n];
     S[0]=0, S[1]=1;
     sort(p.begin(), p.end(),compare);
     double d = norm(p[s]-p[t]);
for (int i=2;i<n;S[m++]=i++)
for(int j=0; j<m; j++){
     if (norm(p[S[j]]-p[i])<d)
          d=norm(p[s=S[j]]-p[t=i]);
     if (real(p[S[j]]) < real(p[i])-d)
          S[j--] = S[--m];
     }
     return make_pair( p[s], p[t] );
}
P rotate(P p1, double a){
double x=p1.real()*cos(a)-p1.imag()*sin(a);
double y=p1.real()*sin(a)+p1.imag()*cos(a);
     return P(x,y);
}
typedef vector <P> Tr;
```

```
Tr make_tr(P a,P b,P c){
  Tr r(3);
  r[0]=a; r[1]=b; r[2]=c;
  return r;
}
bool tr_contains(Tr t,P p){
  return ccw(t[0],t[1],p)>=0 &&
         ccw(t[1],t[2],p)>=0 &&
         ccw(t[2],t[0],p)>=0;
}
bool ear_Q(int i,int j,int k,Pol pol){
 Tr t = make_tr(pol[i], pol[j], pol[k]);
  if (ccw(t[0],t[1],t[2])<=0) return false;
  for (int m=0; m<pol.size(); ++m)
    if (m!=i && m!=j && m!=k)
      if (tr_contains(t, pol[m]))
        return false;
  return true;
}
void triangulate(Pol pol, vector<Tr> &t){
  int n=pol.size();
  vector<int> l, r;
  for (int i=0; i<n; ++i){
    l.push_back((i-1+n)%n);
    r.push_back((i+1+n)%n);
  }
  int i=n-1;
  while (t.size()<n-2){
    i = r[i];
    if (ear_Q(l[i],i,r[i],pol)){
      t.push_back(make_tr(pol[l[i]],pol[i],pol
[r[i]]));
      l[r[i]]=l[i];
      r[l[i]]=r[i];
    }
  }
}
pair<P,P> CCInter(P c1, double r1, P c2,
double r2){
  P A=conj(c2-c1);
  P B=(r2*r2-r1*r1-(c2-c1)*conj(c2-c1)),
C=r1*r1*(c2-c1);
  P D = B*B- 4.0*A*C;
  P z1 = (-B+sqrt(D))/(2.0*A)+c1;
  P z2=(-B-sqrt(D))/(2.0*A)+c1;
  return pair<point, point>(z1,z2);
}
//Geometria 3D
struct P3 {
  double x, y, z;
  P3(double X = 0, double Y = 0, double Z =
0): x(X), y(Y), z(Z) { }
};
struct V3 {
  double x, y, z;
  V3(double X=0, double Y=0, double Z=0):
x(X), y(Y), z(Z) { }
V3(P3 p) { x = p.x; y = p.y; z = p.z; }
V3(P3 p, P3 q) { x = q.x - p.x; y = q.y - p.y;
z = q.z - p.z; }
};
P3 operator + (const P3 &p, const V3 &v){
return P3(p.x+v.x,p.y+v.y,p.z+v.z);}
P3 operator + (const P3 &p, const P3 &q){
  return P3(p.x+q.x,p.y+q.y,p.z+q.z);}
```

```
P3 operator - (const P3 &p, const V3 &v){
 return P3(p.x-v.x, p.y-v.y, p.z-v.z);}
P3 operator - (const P3 &p, const P3 &q){
  return P3(p.x-q.x, p.y-q.y, p.z-q.z);}
V3 operator + (const V3 &u, const V3 &v){
  return V3(u.x+v.x, u.y+v.y, u.z+v.z);}
V3 operator - (const V3 &u, const V3 &v){
  return V3(u.x-v.x, u.y-v.y, u.z-v.z);}
V3 operator * (const double &a, const V3 &v){
  return V3(a*v.x, a*v.y, a*v.z);}
double dot(const V3 u, const V3 v){
  return u.x*v.x+u.y*v.y+u.z*v.z;}
V3 cross(const V3 u, const V3 v){
return V3(u.y*v.z-u.z*v.y,u.z*v.x-
u.x*v.z,u.x*v.y-u.y*v.x);
}
double norma(const V3 v){
 return sqrt(dot(v, v));}
struct recta{
  P3 a, b;
  recta(P3 A, P3 B): a(A), b(B) { }
  recta(P3 P, V3 V): a(P) { b = P + V; }
};
struct semirecta{
  P3 a, b;
  semirecta(P3 A, P3 B): a(A), b(B) { }
  semirecta(P3 P, V3 V): a(P) { b=P+V; }
};
struct segmento {
  P3 a, b;
  segmento(P3 A, P3 B): a(A), b(B) { }
};
```

```
struct triangulo {
  P3 a, b, c;
  triangulo(P3 A,P3 B,P3 C):a(A),b(B),c(C) { }
};
double distancia(const P3 a, const P3 b){
  return norma(V3(a, b));}
double distancia(const P3 p, const recta r){
  V3 v(r.a, r.b), w(r.a, p);
  return norma(cross(v, w)) / norma(v);
}
double distancia(P3 p, semirecta s){
  V3 v(s.a, s.b), w(s.a, p);
  if (dot(v,w)<=0) return distancia(p, s.a);
  return distancia(p, recta(s.a, s.b));
}
double distancia(P3 p, segmento s){
  V3 v(s.a, s.b), w(s.a, p);
  double c1 = dot(v, w), c2 = dot(v, v);
  if (c1 <= 0) return distancia(p, s.a);
  if (c2 <= c1) return distancia(p, s.b);
  return distancia(p, s.a + (c1/c2)*v);
}
double distancia(recta r, recta s){
  V3 u(r.a, r.b), v(s.a, s.b), w(r.a, s.a);
  double a=dot(u,u),b=dot(u,v),c=dot(v,v),
d=dot(u,w),e=dot(v,w);
  double D = a*c - b*b, sc, tc;
  if (D < EPS) {
    sc = 0;
    tc = (b > c) ? d/b : e/c;
  } else {
    sc = (b*e - c*d) / D;
    tc = (a*e - b*d) / D;
  }
  V3 dP = w + (sc * u) - (tc * v);
  return norma(dP);
}
double distancia(segmento r, segmento s){
  V3 u(r.a, r.b), v(s.a, s.b), w(s.a, r.a);
  double a=dot(u,u),b=dot(u,v),c=dot(v,v),
d=dot(u,w),e=dot(v,w);
  double D = a*c - b*b;
  double sc, sN, sD = D;
  double tc, tN, tD = D;
  if (D < EPS) {
  sN = 0;sD = 1;tN = e;tD = c;
  } else {
  sN = (b*e – c*d);
  tN = (a*e - b*d);
   if (sN < 0) {
    sN = 0;tN = e;tD = c;
   } else if (sN > sD) {
     sN = sD;tN = e + b;tD = c;
   }
  }
  if (tN < 0) {
   tN = 0;
  if (-d < 0) {
    sN = 0;
  } else if (-d > a) {
    sN = sD;
  } else {
    sN = -d;
    sD = a;
   }
  } else if (tN > tD) {
    tN = tD;
    if ((-d + b) < 0) {
      sN = 0;
    } else if (-d + b > a) {
      sN = sD;
    } else {
      sN = -d + b;
      sD = a;
     }
  }
  sc = fabs(sN) < EPS ? 0 : sN / sD;
  tc = fabs(tN) < EPS ? 0 : tN / tD;
  V3 dP = w + (sc * u) - (tc * v);
  return norma(dP);
}
V3 projecao(V3 u, V3 v) {
  return (dot(v, u) / dot(u, u)) * u;
}
bool between(P3 a, P3 b, P3 p) {
  return dot(V3(p - a), V3(p - b)) < EPS;
}
double linedist(P3 a, P3 b, P3 p) {
  P3 proj=a+projecao(V3(a, b), V3(a, p));
  if (between(a, b, proj)) {
    return norma(V3(proj, p));
  } else {
    return min(norma(V3(a,p)),norma(V3(b,p)));
  }
}
double distancia(P3 p, triangulo T) {
  V3 X(T.a, T.b), Y(T.a, T.c), P(T.a, p);
  V3 PP = P - projecao(cross(X, Y), P);
```

```
  P3 PPP = T.a + PP;
  V3 R1 = cross(V3(T.a, T.b), V3(T.a, PPP));
  V3 R2 = cross(V3(T.b, T.c), V3(T.b, PPP));
  V3 R3 = cross(V3(T.c, T.a), V3(T.c, PPP));
  if (dot(R1,R2)>-EPS && dot(R2,R3)>-EPS &&
dot(R1,R3)>-EPS){
    return norma(V3(PPP, p));
  } else {
    return min(linedist(T.a,T.b,p),
min(linedist(T.b,T.c,p),linedist(T.c,T.a,p)));
  }
}
//Teoria de numeros
int extGcd(int a, int b, int &x, int &y){
     int g = a; x = 1; y = 0;
     if (b != 0){
          g = extGcd(b, a%b, y, x);
          y -= (a/b)*x;
     }
     return g;
}
bool mExtGcd(int a,int b,int c,int &x,int &y){
     int r = extGcd(a,b,x,y);
     if (c%r != 0) return false;
     x*=c/r; y*=c/r;
     return true;
}
vector<int> primes;
int MAX = 1000000;
//Miller Rabin
typedef unsigned long long u64;
u64 multiply(u64 a, u64 b, u64 mod) {
     u64 rx = 0, sx = 0;
     register int bx;
     for (bx = 0; b >> bx > 0; ++bx) {
         sx += (bx) ? sx : a;
         if (sx >= mod)
             sx -= mod;
         rx += ((b >> bx) & 1) ? sx : 0;
         if (rx >= mod)
             rx -= mod;
     }
     return rx;
}
u64 modpow(u64 a, u64 b, u64 mod) {
     u64 rx = 1, sx = 0;
     register int bx;
     for (bx = 0; b >> bx > 0; ++bx) {
         sx = (bx)?multiply(sx, sx, mod) : a;
  rx = ((b>>bx)&1)?multiply(rx, sx, mod) : rx;
     }
     return rx;
}
u64 f(u64 x, u64 mod) {
     u64 rx = multiply(x, x, mod) + 123;
     while (rx >= mod)
         rx -= mod;
     return rx ? rx : 2;
}
bool miller_rabin(u64 n, int iter) {
     u64 m = n - 1, b = 2, z;
     register int i, j, a = 0;
     while (!(m & 1)) {
         m >>= 1;
         ++a;
     }
```

```
     for (i = 0; i < iter; ++i) {
         j = 0, z = modpow(b, m, n);
         while (!((j==0&&z==1)||z==n-1)) {
             if ((j>0 && z == 1) || ++j == a)
                 return false;
             z = modpow(z, 2, n);
         }
         b = f(b, n);
     }
     return true;
}
bool is_prime(u64 n) {
     return n == 2 ||
(n > 1 && (n & 1) && miller_rabin(n, 10));
}
int josephus(int n, int k){
     int f = 0;
     for (int i=0; i<n; i++) f = (f+k)%(i+1);
     return f+1;
}
//Inverso multiplicativo a*inv == 1 (mod m)
bool invMult(long long a, long long m, long
long &inv) {
     long long x, y, r;
     r = extGcd(a, m, x, y);
     if (r!=1) return false;
     inv = x;
     if (inv<0) inv += m;
     return true;
}
//a*x == b (mod n)
bool MLE(long long a, long long b, long long
n, long long &x){
    long long d, xx, y;
    d = extGcd(a,n,xx,y);
    if (b%d) return false;
    x = ((xx*(b/d))%n+n)%n;
    return true;
}
// teorema del resto chino x == r[i] (mod m[i])
bool TRC (vector<long long> r, vector<long long> m, long long &x, long long &M){
    int n=r.size();
    long long inv;
    x=0; M=1;
    for (int i=0; i<n; i++) M*=m[i];
    for (int i=0; i<n; i++){
        if (!invMult(M/m[i],m[i],inv)) return false;
        x+=r[i]*(M/m[i])*inv;
    }
    x = (x%M);
    return true;
}
```

//Euler’s totient theorem

If n is a positive integer and a is coprime to n, then a^phi(n) == 1 (mod n)

//Teorema de Wilson

Si p es un número primo, entonces (p–1)! == -1 mod(p).

//Fermat's little theorem

If p is a prime number, then for any integer a that is coprime to p, we have a^p ≡ a (mod p)

//Discrete logarithm theorem

Si g es una raiz primitiva de Zn entonces la ecuacion g^x == g^y (mod n) se cumple si y solo si se cumple x == y mod(phi(n)).

g es una raiz primitiva mod n si las potencias de g modulo n van por todos los coprimos de n.

La raiz primitiva existe si n = 2, 4, p^k o 2*p^k donde p es un primo impar.

Para comprbar que g es una raiz primitiva de n solo tenemos q comprobar que g^d != 1 mod(n) para todo primo p que divide a phi(n), d = phi(n)/p.

//Cantidad de digitos de n!

```
(long long)floor( (log(2*acos(-1)*a)/2 + a*(log(a)-1) ) /log(10) ) + 1);
```

//Probabilidad

P (E1 ∪ E2) + P (E1 ∩ E2) = P (E1) + P (E2). Entonces si E1 y E2 son mutuamente exclusivos, P (E1 ∪ E2) = P (E1) + P (E2).

Probabilidad de que ocurra el evento E1 dado que ha ocurrido el evento E2

P (E1 |E2) = P (E1 ∩ E2)/P (E2)

//Teorema de Bayes

P (E1 |E2) = P (E1)*P (E2 |E1)/P (E2)

//Bernoulli

Una prueba de Bernoulli es aquella que puede terner 2 resultados exito o fallo. Si la probabilidad de exito de una prueba de Bernoulli es p, la probabilidad de q ocurran k exitos en una secuencia de n eventos idependientes es: C(n,k)*(p^k)*(1-p)^(n-k).

//m^(n)

m^(n) = m(m - 1)(m - 2) · · · (m - n + 1).

//Stirling number of the second kind

{m,n} = (1/n!) *Σ(-1)^(n-k)*(n,k)*(k^m)

//Classical occupancy problem

En una urna con m bolas numeradas de 1 a m. Suponga que extraemos n bolas una por una, con remplazamientos. La probabilidad de que hallan sido extraidas exactamente t bolas diferentes es:

P1(m,n,t) = {n,t}*(m^(t))/(m^n) .

//Problema del cumpleanno

En una urna con m bolas numeradas de 1 a m. Suponga que extraemos n bolas una por una, con remplazamientos. La probabilidad de que halla una coincidencia es:

P2(m,n) = 1 – P1(m,n,n) = 1-(m^(n))/(m^n) ≈ 1-exp(-(n*n)/(2*m)). exp(x) = e^x.

Si sacamos n1 bolas de una urna y n2 bolas de otra con remplaso, la probabilidad de coincidencia es:

P3(m,n1,n2) = 1-(1/m^(n1+n2))* Σ(m^(t1+t2)*{n1,t1}*{n2,t2}) ≈ 1-exp(-(n*n)/m).

Si sacamos n1 bolas de una urna y n2 bolas de otra sin remplaso, la probabilidad de coincidencia es:

P4(m,n1,n2) = 1 – (m^((n1+n2)))/(m^(n1)+m^(n2)).

Si sacamos n1 bolas de una urna con remplazo y n2 bolas de otra sin remplaso, la probabilidad de coincidencia es:

P5(m,n1,n2) = 1-(1-n2/m)^n1.

```
//MAXFLOW
struct Ar{
     int ini , fin ,next, peso;
     Ar(){}
}A[60005];
int n, a,emp[5005],last[5005];
void Read(){
     int x,y,z,r=0;
     scanf("%d%d",&n,&a);
     for(int i = 1 ; i <= n ; i++) emp[i] = -1;
     for(int i = 0 ; i < 2*a ; i+=2){
          scanf("%d%d%d",&x,&y,&z);
          A[r++] = Ar(x,y,emp[x],z);
          emp[x] = r-1;
          A[r++] = Ar(y,x,emp[y],z);
          emp[y] = r-1;
     }
}
int H[5005];
bool Cogi[5005];
bool BFS(){
     queue<int> cola;
     memset(H,-1,sizeof(H));
     H[1]=0;
     cola.push(1);
     while(cola.size() != 0){
          int v = cola.front();
          cola.pop();
          for(int i = emp[v];i != -1; i = A[i].next)
               if(H[A[i].fin]==-1 && A[i].peso != 0){
               cola.push(A[i].fin);
               H[A[i].fin] = H[A[i].ini] + 1;
          }
     }
     return H[n]!=-1;
```

```
}
int DFS_Num[5005],id;
int DFS(int ini ,int flow){
     DFS_Num[ini]= id;
     if(ini == n)return flow;
     for(last[ini]=last[ini]==-1?
emp[ini]:A[last[ini]].next ; last[ini] != -1;
last[ini] = A[last[ini]].next){
          int i = last[ini];
          if(DFS_Num[A[i].fin] != id &&
A[i].peso != 0 && H[A[i].ini]+1 ==
H[A[i].fin]){
          int
k=DFS(A[i].fin,minimo(flow,A[i].peso));
          if(k != 0){
               A[i].peso-=k;
               A[i^1].peso+=k;
               return k;
          }
          }
     }
     return 0;
}
void Flow(){
     long long int flow = 0;
     int k = 0;
     while(BFS()){
          memset(last,-1,4*n+4);
          while(k = DFS(1,1000000001),k){
               flow+=k;
               id++;
          }
          id++;
     }
     printf("%lld\n",flow);
}
int main(){    id = 1;Read();Flow();}
//MINIMAL ASSIGMENT TRANSPORT PROBLEM
int w[1000][1000],r[1000][1000];
int cx[1000],cy[1000],n,m;
int lx[1000],ly[1000],vx[1000],vy[1000];
int slack[1000],slackx[1000],Enl[1000];
int t,i,j,k,u,bot,delta,ans;
bool found;
int Hung(){
  for(u=0;u<n;u++)
  while(cx[u]){
    for(i=0;i<n;i++){ vx[i]=0; Enl[i]=-1; }
    for(i=0;i<m;i++){
      vy[i]=0;
      slack[i]=lx[u]+ly[i]-w[u][i];
     slackx[i]=u;
    }
    vx[u]=1;
    while(1){
    delta=0x7fffffff;
    found=false;
    for(i=0;i<m;i++)
      if(!vy[i]){
       delta=min(slack[i],delta);
       if(slack[i]==0){
         vy[i]=1;
         if(cy[i]){
          bot = min(cx[u],cy[i]);
          for(j=slackx[i];Enl[j]!=-
1;j=slackx[Enl[j]])
```

```
            bot=min(bot,r[j][Enl[j]]);
          cx[u]-=bot;
          cy[i]-=bot;
          for(j=i;j!=-1;j=Enl[slackx[j]]){
            r[slackx[j]][j]+=bot;
            if(Enl[slackx[j]]!=-1)
            r[slackx[j]][Enl[slackx[j]]]-=bot;
          }
          found=true;
       }else{
           for(j=0;j<n;j++)
             if(!vx[j]&&r[j][i]){
               Enl[j]=i;
               vx[j]=1;
               for(k=0;k<m;k++)
          if(!vy[k]&&slack[k]>lx[j]+ly[k]-w[j]
[k]){
                slack[k]=lx[j]+ly[k]-w[j][k];
                slackx[k]=j;
               }
             }
          }
         break;
    }
   }
 if(found)break;
  if(delta){
     for(i=0;i<n;i++)
      if(vx[i])
       lx[i]-=delta;
       for(i=0;i<m;i++){
          if(vy[i]) ly[i]+=delta;else
        slack[i]-=delta;
```

```
       }
  }
 }
}
ans=0;
for(i=0;i<n;i++)
 for(j=0;j<m;j++)
  {
     ans-=r[i][j]*w[i][j];
     /*if(r[i][j])
     cout<<i+1<<"-->"<<j+1<<endl;*/
  }
return ans;
}
int main(){
 scanf("%d",&t);
 while(t--){
  scanf("%d%d",&n,&m);
  for(i=0;i<n;i++){
     scanf("%d",cx+i);
     lx[i]=-0x80000000;
  }
  for(i=0;i<m;i++){
     scanf("%d",cy+i);
     ly[i]=0;
  }
  for(i=0;i<n;i++)
   for(j=0;j<m;j++){
      scanf("%d",&w[i][j]);
      r[i][j]=0;
      w[i][j]=-w[i][j];
      lx[i]=max(w[i][j],lx[i]);
   }
```

```
     printf("%d ",Hung());
 }
 return 0;
}
//Method: Finding the Kth Shortest Path
#define for_each(it, v) for
(vector<Edge*>::iterator it = (v).begin();
it != (v).end(); ++it)
const int MAX_N = 10000;
const int MAX_M = 50000;
const int MAX_K = 10000;
const int INF = 1000000000;
struct Edge{
     int from, to;
     int weight;
};
struct HeapNode{
     Edge* edge;
     int depth;
     HeapNode* child[4];
};
int n, m, k, s, t;
Edge* edge[MAX_M];
int dist[MAX_N];
Edge* prev[MAX_N];
vector<Edge*> graph[MAX_N];
vector<Edge*> graphR[MAX_N];
HeapNode* nullNode;
HeapNode* heapTop[MAX_N];
HeapNode* createHeap(HeapNode* curNode,
HeapNode* newNode){
     if (curNode == nullNode)
          return newNode;
     HeapNode* rootNode = new HeapNode;
     memcpy(rootNode, curNode,
sizeof(HeapNode));
     if (newNode->edge->weight<curNode->edge-
>weight){
          rootNode->edge = newNode->edge;
          rootNode->child[2] = newNode-
>child[2];
          rootNode->child[3] = newNode-
>child[3];
          newNode->edge = curNode->edge;
          newNode->child[2] = curNode-
>child[2];
          newNode->child[3] = curNode-
>child[3];
     }
     if (rootNode->child[0]->depth<rootNode-
>child[1]->depth)
          rootNode->child[0] =
createHeap(rootNode->child[0], newNode);
     else
          rootNode->child[1] =
createHeap(rootNode->child[1], newNode);
     rootNode->depth = max(rootNode->child[0]-
>depth, rootNode->child[1]->depth) + 1;
     return rootNode;
}
bool heapNodeMoreThan(HeapNode* node1,
HeapNode* node2){
     return node1->edge->weight > node2->edge-
>weight;
}
int main(){
 scanf("%d%d%d", &n,&m,&k);scanf("%d%d",
&s,&t);
 s--, t--;
 while (m--){
     Edge* newEdge = new Edge;
     int i, j, w;
     scanf("%d%d%d", &i, &j, &w);
     i--, j--;
     newEdge->from = i;
     newEdge->to = j;
     newEdge->weight = w;
     graph[i].push_back(newEdge);
     graphR[j].push_back(newEdge);
  }
     //Dijkstra
     queue<int> dfsOrder;
     memset(dist, -1, sizeof(dist));
     typedef pair<int, pair<int, Edge*> >
DijkstraQueueItem;
     priority_queue<DijkstraQueueItem,
vector<DijkstraQueueItem>,
greater<DijkstraQueueItem> > dq;
     dq.push(make_pair(0, make_pair(t, (Edge*)
NULL)));
     while (!dq.empty()){
      int d = dq.top().first;
      int i = dq.top().second.first;
      Edge* edge = dq.top().second.second;
      dq.pop();
      if (dist[i] != -1) continue;
      dist[i] = d;prev[i] = edge;
      dfsOrder.push(i);
      for_each(it, graphR[i])
```

```
      dq.push(make_pair(d + (*it)->weight,
make_pair((*it)->from, *it)));
     }
//Create edge heap
    nullNode = new HeapNode;
     nullNode->depth = 0;
     nullNode->edge = new Edge;
     nullNode->edge->weight = INF;
     fill(nullNode->child, nullNode->child + 4,
nullNode);
     while (!dfsOrder.empty()){
          int i = dfsOrder.front();
          dfsOrder.pop();
          if (prev[i] == NULL) heapTop[i] =
nullNode;
          else
               heapTop[i] = heapTop[prev[i]-
>to];
          vector<HeapNode*> heapNodeList;
          for_each(it, graph[i])
          {
               int j = (*it)->to;
               if (dist[j] == -1)continue;
               (*it)->weight += dist[j] -
dist[i];
               if (prev[i] != *it){
               HeapNode* curNode = new
HeapNode;
     fill(curNode->child, curNode->child+4,
nullNode);
                    curNode->depth = 1;
                    curNode->edge = *it;
     heapNodeList.push_back(curNode);
              }
         }
         if (!heapNodeList.empty()){
              make_heap(heapNodeList.begin(),
heapNodeList.end(), heapNodeMoreThan);
              int size = heapNodeList.size();
              for (int p = 0; p<size; p++)  {
heapNodeList[p]->child[2] = 2*p+1<size?
heapNodeList[2 * p+1]:nullNode;
heapNodeList[p]->child[3] = 2*p+2<size?
heapNodeList[2 * p+2]:nullNode;
              }
heapTop[i]=createHeap(heapTop[i],
heapNodeList.front());
         }
     }
     //Walk on DAG
     typedef pair<long long, HeapNode*>
DAGQueueItem;
     priority_queue<DAGQueueItem,
vector<DAGQueueItem>, greater<DAGQueueItem> >
aq;
     if (dist[s] == -1) printf("NO ");
     else{
       printf("%d ", dist[s]);
       if (heapTop[s] != nullNode)
          aq.push(make_pair(dist[s]+heapTop[s]-
>edge->weight, heapTop[s]));
     }
     k--;
     while (k--)    {
          if (aq.empty()){
```

```
                    printf("NO ");
                    continue;
               }
               long long d = aq.top().first;
               HeapNode* curNode = aq.top().second;
               aq.pop();
               printf("%lld\n",d);
               if (heapTop[curNode->edge->to]!
=nullNode)
                    aq.push(make_pair(d +
heapTop[curNode->edge->to]->edge->weight,
heapTop[curNode->edge->to]));
               for (int i = 0; i < 4; i++)
                    if (curNode->child[i] !=
nullNode)
                    aq.push(make_pair(d - curNode-
>edge->weight + curNode->child[i]->edge-
>weight, curNode->child[i]));
     }
}
//MaxflowMincost
struct Edge{
  int u , v , cap , next; long long  cost;
  Edge(){}
}A[1000];
int total,L[5000],n,m,s,r,x,y;
long long cost;
void ADD(int u,int v,int cap,long long cost){
  A[total] = Edge(u,v,cap,cost,L[u]);
  L[u] = total++;
  A[total] = Edge(v,u,0,-cost,L[v]);
  L[v] = total++;
}
```

```
int Flow[30000],fl;
long long Dist[5000],Phi[5000],Prev[5000];
bool In[5000];
struct Node{
     int u; long long peso;
     Node(){}
     bool operator <(Node const &l)const{
          return peso > l.peso;
     }
}; priority_queue<Node> cola;
bool Dikjstra(){
   for(int i = 0 ; i < 2*n ; i++){
        Flow[i]=In[i]=0; Dist[i]=INF;
   }
   cola.push(Node(s,0));
   In[s] = Dist[s] = 0; Flow[s] = INF;
   while(cola.size()){
        x = cola.top().u;
        cost = cola.top().peso;
        cola.pop();
        fl = Flow[x];
        if(In[x]) continue;
        In[x] = 1;
        for(int i=L[x] ; i!=-1 ; i=A[i].next){
             y = A[i].v;
             if(A[i].cap>0 && (Dist[y]>
cost+A[i].cost+Phi[x]-Phi[y])){
             Dist[y] = cost+A[i].cost+Phi[x]-
Phi[y];
             Flow[y] = min(fl,A[i].cap);
             cola.push( Node(y,Dist[y]) );
             Prev[y] = i;
             }
        }
   }
   return Flow[r]!=0;
}
long long MAX_FLOW_MIN_COST(){
     long long cost = 0;
     int fl = 0;
   while(Dikjstra()){
        cost += (Dist[r]+Phi[r])*Flow[r];
        fl+=Flow[r];
        x = r;
        for(int i = 0 ; i <= 2*n ; i++)
             if(Flow[i])
                Phi[i]+=Dist[i];
        while(x != s){
             A[Prev[x]].cap-=Flow[r];
             A[Prev[x]^1].cap+=Flow[r];
             x = A[Prev[x]].u;
        }
   }
     return cost;
}
int main(){
   while(scanf("%d%d",&n,&m),n+m){
      s=0; r=n-1;
      while(m--){
           scanf("%d%d%lld",&x,&y,&cost);
           ADD(x,y,1,cost);
      }
      memset(L,-1,sizeof(L));
      long long sol = MAX_FLOW_MIN_COST();
      printf("%lld ",sol);
   }
```

```
 return 0;
}
//BIA
struct node{
  vector<int> adj;
};
const int MAX = 5005;
vector<node> t;
vector<node> inver;
int x,y,n,m;
int dfnumber[MAX],level[2* MAX],e[2* MAX],c;
int father[5005],first[MAX],cant,sp[2* MAX]
[20],log[2* MAX],f[MAX],r[MAX],low[MAX];
bool use[MAX],mark[MAX];
vector<int> vis;
void dfs(int nodo,int lvl){
  vis.push_back(nodo);
  use[nodo]=1;
  dfnumber[nodo] = c++;
  level[cant] = lvl;
  e[cant] = nodo;
  if(first[nodo]==-1) first[nodo] = cant;
  cant++;
  for(int i=0;i<t[nodo].adj.size();++i)
   if(!use[t[nodo].adj[i]]){
        father[t[nodo].adj[i]]=nodo;
        dfs(t[nodo].adj[i],lvl+1);
        level[cant]=lvl;
        e[cant]=nodo;
        cant++;
   }
}
int query(int x,int y){
  int ini = first[x],fin = first[y],aux;
  if(ini > fin){
      aux = ini;
      ini = fin;
      fin = aux;
   }
   int k = log[fin-ini+1];
   if(level[sp[ini][k]] < level[sp[fin-(1<<k)
+1][k]])
   return e[sp[ini][k]];
   return e[sp[fin-(1<<k)+1][k]];
}
void Dp(){
   for(int i=0;i<cant;i++)
     sp[i][0] = i;
    for(int j=1;(1<<j)<=cant;j++)
     for(int i=0;i+(1<<j)<cant;i++)
      if(level[sp[i][j-1]]<level[sp[i+(1<<(j-
1))][j-1]])
       sp[i][j] = sp[i][j-1];
      else
       sp[i][j] = sp[i+(1<<(j-1))][j-1];
}
int find(int x){
  if(f[f[x]]==f[x])
     return f[x];
  int tmp=f[x];
  f[x]=find(f[x]);
  r[x]=min(r[x],r[tmp]);
   return f[x];
}
void Compute_Dominators(){
     for(int i=0;i<=n;i++){
          f[i]=i;
          low[i] = dfnumber[i];
     }
     fill(mark,mark+n+1,0);
     while(!vis.empty()){
          int u=vis.back(),v;
          for(int i=0;i<inver[u].adj.size();i+
+){
               v= inver[u].adj[i];
               if(v==father[u])continue;
               if(!
mark[v])low[u]=min(low[u],low[v]);
               else{
               int lca=query(u,v);
               if(lca!
=u)low[u]=min(low[u],low[lca]);
               find(v);
               low[u]=min(low[u],r[v]);
          }
          }
          mark[u]=true;
          f[u]=father[u];
          r[u]=low[u];
          vis.pop_back();
     }
}
int main(){
   log[0]=log[1]=0;
   int pot = 2;
   for(int i=2;i<=10001;i++){
          log[i] = log[i-1];
          if(i==pot){
            log[i]++;
```

```
       pot*=2;
      }
  }
int cas = 10;
while(cas--){
    t.clear();
    inver.clear();
     scanf("%d%d",&n,&m);
     t.resize(n+1);
     inver.resize(n+1);
     vis.clear();
  while(m--){
       scanf("%d%d",&x,&y);x--;y--;
       t[x].adj.push_back(y);
       inver[y].adj.push_back(x);
  }
  c=0;
  cant = 0;
  memset(use,0,sizeof(use));
  fill(first,first+n+1,-1);
  dfs(0,0);
  Dp();
  Compute_Dominators();
  memset(mark,0,sizeof(mark));
     int sol=0;
     for(int i=1;i<n;i++)
          if(low[i]>=dfnumber[father[i]])
               mark[father[i]]=true;
     for(int i=0;i<n;i++)
          if(mark[i])sol++;
     printf("%d\n1",sol);
     for(int i=1;i<n;i++)
          if(mark[i])printf(" %d",i+1);
```

```
        printf("\n");
     }
   return 0;
}
//Edmond’s
#define maxN 300
int n,match[maxN],Head,
     Tail,Queue[maxN],Start,
     Finish,NewBase,Father[maxN],Base[maxN],Cou
     nt;
bool graph[maxN][maxN],InQueue[maxN],
     InPath[maxN], InBlossom[maxN];

void CreateGraph(){
  int u,v;
  memset(graph,0,sizeof(graph));
  scanf("%d",&n);
  while(scanf("%d%d",&u,&v)!=EOF){
          graph[u][v]=graph[v][u]=1;
  }
}
void Push(int u){Queue[Tail++]= u; InQueue[u]=
     true;}
int Pop(){ return Queue[Head++]; }
int FindCommonAncestor(int u, int v){
        memset(InPath, 0, sizeof(InPath));
        while(true){
          u=Base[u]; InPath[u]= true;
          if (u==Start)break;
          u= Father[match[u]];
        }
        while(true){
          v= Base[v];
          if (InPath[v]) break;
          v = Father[match[v]];
        }
       return v;
}
void ResetTrace(int u){
   int v;
   while (Base[u] != NewBase){
       v= match[u];
       InBlossom[Base[u]]= 1;
       InBlossom[Base[v]]= 1;
       u= Father[v];
       if (Base[u] != NewBase)Father[u]=v;
     }
}
void  BlossomContract(int u,int  v){
   NewBase= FindCommonAncestor(u, v);
   memset(InBlossom,0 ,sizeof(InBlossom));
   ResetTrace(u);
   ResetTrace(v);
   if (Base[u] != NewBase)Father[u]= v;
   if (Base[v] != NewBase)Father[v]= u;
   for(u=1;u<=n;u++)
     if (InBlossom[Base[u]]){
        Base[u]= NewBase;
        if (!InQueue[u]) Push(u);
      }
}
void FindAugmentingPath(){
    int u,v;
    memset(InQueue,false, sizeof(InQueue));
    memset(Father,0,sizeof(Father));
    for(u=1;u<=n;u++) Base[u]=u;
    Head= Tail= 1; Push(Start); Finish = 0;
    while (Head < Tail) {
       u= Pop();
       for (v=1;v<=n;v++)
     if ((graph[u][v])&&(Base[u]!
     =Base[v])&&(match[u]!= v))
         if ((v==Start)||
     ((match[v]>0)&&(Father[match[v]] > 0)))
            BlossomContract(u, v);
            else if (Father[v] == 0){
              Father[v]=u;
              if (match[v] > 0)
               Push(match[v]);
              else{
                Finish=v;
                return;
              }
            }
     }
}
void AugmentPath(){
        int u,v,w;
    u=Finish;
    while(u > 0){
       v=Father[u];
       w=match[v];
       match[v]= u;
       match[u]= v;
       u= w;
     }
}
void Edmonds(){
 int u;
```

```
 memset(match,0,sizeof(match));
  for(u=1;u<=n;u++)
   if (match[u]==0){
      Start=u;
      FindAugmentingPath();
      if (Finish > 0) AugmentPath();
    }
}
void PrintMatch(){
 int u;
 for(u=1;u<=n;u++)
  if (match[u] > 0) Count++;
    printf("%d\n",Count);
  for(u=1;u<=n;u++)
     if (u < match[u])printf("%d
     %d\n",u,match[u]);
}
int main(){
  CreateGraph();
  Edmonds();
  PrintMatch();
  return 0;
}
```