```
/*/Microhard Team Code Book
Codebook--- Number Theory ----------------------- p ___
        |   |
        |   --- Square Country Problem
        |   |
        |   --- Extended GCD
        |   |
        |   --- Linear Seive for Prime
        |   |
        |   --- Fast Computation for Exp
        |   |
        |   --- phi funtion
        |   |
        |   --- Narayana number
        |   |
        |   --- Mersenne prime
        |   |
        |   --- Fibonacci
        |
        --- Combinatorial geometry --------------- p ___
        |   |
        |   --- 2D Convex Hull
        |   |
        |   --- 3D Convex Hull
        |
        --- Graph Theory
        |   |
        |   ------- Min Cost Flow ---------------- p ___
        |   |
        |   ------- Min Cost Flow fast ----------- p ___
        |   |
        |   ------- Topological Sort ------------- p ___
        |   |
        |   ------- Strongly Connected Component - p ___
        |   |
        |   ------- RMQ --------------------------- p ___
        |   |
        |   ------- LCA  -------------------------- p ___
        |   |
        |   ------- Bridge ------------------------ p ___
        |   |
        |   ------- Biconnected Component -------- p ___
        |   |
        |   ------- Articulation Point ----------- p ___
        |   |
        |   ------- Prim -------------------------- p ___
        |   |
        |   ------- Kruskal ----------------------- p ___
        |   |
        |   ------- Min Edge-disjoint Path Cover - p ___
        |
        --- Data Structure
        |   |
        |   ------- RB Tree ----------------------- p ___
        |   |
        |   ------- Binary Indexed Tree ---------- p ___
        |
        |--- String
        |   |
        |   ------- KMP ---------------------------- p ___
        |   |
        |   ------- Multi-matching --------------- p ___
        |   |
        |   ------- LIS ---------------------------- p ___
        |   |
        |   ------- LIS - STL --------------------- p ___
        |
        |--- Others
            |
            ------- Joseph's Problem ------------- P ___
            |
            ------- Big Num ---------------------- p ___
            |
            ------- Difference_constraints ------- p ___
*/
```

```
///////*****   Square Country Problem   *******///////
initialize: do q[i^2] = 1;
            and do if (q[i^2+j^2] ==0) q[i^2+j^2] = 2;
// Note: q[5^2]=1 not 2
if(q[n]>0)       ans=q[n];
if(n%8==4)       ans=4;
while(n>0&&n%4==0){  n/=4; if(n%8==7) ans=4 }
                 ans=3 //not case above
---
///////*******   Extended GCD   *******///////
typedef long long LL;
// Ax+By=gcd(A,B)
//This function returns an x
LL ext_gcd(LL A, LL B, LL p, LL q, LL r, LL s) {
  if (B == 0) return p;
  return ext_gcd(B, A%B, r, s, p-r*(A/B), q-s*(A/B));
}
---
///////*******   Linear Seive for Prime *******///////
#define MAXNUM 32772 // count to MAXNUM-1
int primes[MAXNUM];
int findPrimes(void){
    int a[MAXNUM+1],num=0,i,j;
    memset(a, 0, sizeof(a));
    for (i = 2; i <= MAXNUM; ++i){//can erase the even number first, i=3 and i+=2
                                  //be careful overflow of (i*i)
        if (!a[i]) //i is a prime
            primes[num++]=i;
        for(j=0;primes[j]<=i && primes[j]*i<= MAXNUM;j++){
            a[primes[j] * i] = 1; //mark for not a prime
            if (i % primes[j] == 0)  break;
        }
    }
    return num;
}
---
///////*******   Fast Computation for Exp   *******///////
typedef long long LL;
LL bigmod(LL g, LL h, LL n){
    if(h==0)    return 1LL;
    LL a = (g*g)%n;
    return bigmod(a,h/2,n) * (h%2?g,1LL) % n;
}
---
///////******* phi funtion *******///////
#define MAXNUM 30
int prime[MAXNUM],phi[MAXNUM];
void find_phi(void){
    int num=0,i,j;
    memset(phi,0,sizeof(phi));// phi[] will be a flag as will
    for(i=2;i<MAXNUM;i++){
        if(phi[i]==0)   phi[i]=i-1,prime[num++]=i;//prime's phi = p-1
        for(j=0;j<num && prime[j]*i<=MAXNUM;j++){
            if(i%prime[j]==0){
                phi[prime[j]*i] = phi[i]* prime[j];
                break;
            }
            phi[prime[j]*i] = phi[i] * (prime[j]-1);// because (prime[j],i)=1
        }
    }
}
---
///////******* NTUJudge 0391 - Narayana number *******///////
N(n,k) = (1/n) * C(n,k) * C(n,k-1) (mod 32771)
1<=k<=n<=10^9
 C(m,n) = C(m/p,n/p) * C(m%p,n%p) (mod p)
---
///////******* Mersenne prime  *******///////
M_p = 2^m - 1 is prime
2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281,...
---
///////******* Fibonacci *******///////
#include <iostream>
using namespace std;
void F_mult(int n , long long m[], int p);
int F(int n, int p)   //F[n] % p
{
    if(n == 1) return 1%p;
    else if(n == 2) return 1%p;
    else
    {
        long long tmp[4];
        F_mult(n-2, tmp, p);
        return ((tmp[0]%p) + (tmp[1]%p))%p;
    }
}
void F_mult(int n , long long m[], int p){
    if(n > 1){
        long long tmp[4];
        F_mult(n/2, tmp, p);
        if(n % 2 == 0){
            m[0] = (tmp[0]%p)*(tmp[0]%p) + (tmp[1]%p)*(tmp[2]%p);
            m[1] = (tmp[0]%p)*(tmp[1]%p) + (tmp[1]%p)*(tmp[3]%p);
            m[2] = (tmp[2]%p)*(tmp[0]%p) + (tmp[3]%p)*(tmp[2]%p);
            m[3] = (tmp[2]%p)*(tmp[1]%p) + (tmp[3]%p)*(tmp[3]%p);
        }
        else{
            m[0] = (tmp[0]%p)*(tmp[0]%p) + (tmp[1]%p)*(tmp[2]%p)
```

```
                + (tmp[2]%p)*(tmp[0]%p) + (tmp[3]%p)*(tmp[2]%p);
            m[1] = (tmp[0]%p)*(tmp[1]%p) + (tmp[1]%p)*(tmp[3]%p)
                + (tmp[2]%p)*(tmp[1]%p) + (tmp[3]%p)*(tmp[3]%p);
            m[2] = (tmp[0]%p)*(tmp[0]%p) + (tmp[1]%p)*(tmp[2]%p);
            m[3] = (tmp[2]%p)*(tmp[1]%p) + (tmp[3]%p)*(tmp[3]%p);
        }
    }
    else if(n == 1){
        m[0] = 1; m[1] = 1; m[2] = 1; m[3] = 0;
    }else{    //n == 0
        m[0] = 1; m[1] = 0; m[2] = 0; m[3] = 1;
    }
    m[0] %= p;  m[1] %= p;  m[2] %= p; m[3] %= p;
}
---
///////******* 2D-Convex Hull *******///////
#include "iostream"
#include "math.h"
#include "cmath"
using namespace std;
typedef struct vector{
        double x,y;
        double operator^(vector t){
                return x*t.y-y*t.x;
        }
        double operator*(vector t){
                return x*t.x+y*t.y;
        }
};
typedef struct point{
        double x,y,s;
        vector operator-(point t){
                vector tmp;
                tmp.x=x-t.x;    tmp.y=y-t.y;
                return tmp;
        }
        bool operator<(point t) const{
                return s<t.s;
        }
        void operator=(point t){
                x=t.x;  y=t.y;  s=t.s;
        }
};
int q[200001];
int next[100001]={0};
point p[100001];
int f=0,n,temp;
double d=999999999;
point m;
bool check(int a,int b,int c){
        if(((p[b]-p[c])^(p[a]-p[b]))<0) return true;
        if(((p[b]-p[c])^(p[a]-p[b]))==0)
                if(((p[b]-p[c])*(p[a]-p[b]))<0) return true;
        return false;
}
int main(){
        int i,j;
        scanf("%d",&n);
        for(i=0;i<n;i++){
                scanf("%lf %lf",&p[i].x,&p[i].y);
                if(p[i].y<d){
                        d=p[i].y;       temp=i;
                }
        }
        m=p[temp];      p[temp]=p[0];   p[0]=m;
        for(i=1;i<n;i++)
                p[i].s=atan2(p[i].y-p[0].y,p[i].x-p[0].x);
        sort(p+1,p+n);
        p[n]=p[0];
        f=0;    q[f]=0; q[++f]=1;
        for(i=2;i<=n;i++){
                q[++f]=i;
                while(check(q[f],q[f-1],q[f-2]))
                        q[f-1]=q[f--];
        }
        printf("%d\n",f);
        system("pause");
        return 0;
}
---
///////******* 3D - convex *******///////
#include<stdio.h>
#include<math.h>
#include<algorithm>
#define eps 1e-7
#define MAXV 305
#define MAXF 2440     //MAXV*8
using namespace std;
struct pt{
        double x, y, z;
        pt(){}
        pt(double u,double v,double w): x(u), y(v), z(w){}
        pt operator - (pt p){return pt(x-p.x,y-p.y,z-p.z);}
        pt operator * (pt p){return pt(y*p.z-z*p.y, z*p.x-x*p.z, x*p.y-y*p.x);}
        double operator ^ (pt p){return x*p.x+y*p.y+z*p.z;}
};
struct _3DCH{
        struct fac{
```

```
        int a,b,c;
        bool ok;
    };
    int n;
    int cnt;
    pt P[MAXV];
    fac F[MAXF];
    int to[MAXV][MAXV];
    double vlen(pt a){return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}
    double area(pt a,pt b,pt c){return vlen((b-a)*(c-a));}
    double volume(pt a,pt b,pt c,pt d){return (b-a)*(c-a)^(d-a);}
    double ptof(pt &p, fac &f){
        pt m=P[f.b]-P[f.a], n=P[f.c]-P[f.a],t=p-P[f.a];
        return (m*n)^t;
    }
    void deal(int p, int a, int b){
        int f=to[a][b];
        fac add;
        if (F[f].ok){
            if (ptof(P[p],F[f])>eps) dfs(p, f);
            else{
                add.a=b,add.b=a, add.c=p,add.ok=1;
                to[p][b]=to[a][p]=to[b][a]=cnt;
                F[cnt++]=add;
            }
        }
    }
    void dfs(int p,int cur){
        F[cur].ok=false;
        deal(p,F[cur].b,F[cur].a);
        deal(p,F[cur].c,F[cur].b);
        deal(p,F[cur].a,F[cur].c);
    }
    bool same(int s,int t){
        pt &a=P[F[s].a], &b=P[F[s].b],&c=P[F[s].c];
        return fabs(volume(a,b,c,P[F[t].a]))<eps &&
fabs(volume(a,b,c,P[F[t].b]))<eps && fabs(volume(a,b,c,P[F[t].c]))<eps;
    }
    void construct(){
        cnt=0;
        if (n<4) return;
        //for existence of four point on a plane
        bool sb=true;
        for(int i=1;i<n; i++)
            if(vlen(P[0]-P[i])>eps){
                swap(P[1],P[i]);
                sb=false; break;
            }
        if(sb) return;
        sb=true;
        for(int i=2;i<n;i++)
            if(vlen((P[0]-P[1])*(P[1]-P[i]))>eps){
                swap(P[2],P[i]);
                sb=false; break;
            }
        if(sb)return;
        sb=true;
        for(int i=3;i<n;i++)
            if(fabs((P[0]-P[1])*(P[1]-P[2])^(P[0]-P[i]))>eps){
                swap(P[3],P[i]);
                sb=false;
                break;
            }
        if(sb)return;
        //end

        fac add;
        for(int i=0;i<4;i++){
            add.a=(i+1)%4, add.b=(i+2)%4, add.c=(i+3)%4, add.ok=1;
            if(ptof(P[i],add)>0) swap(add.b,add.c);
            to[add.a][add.b]=to[add.b][add.c]=to[add.c][add.a]=cnt;
            F[cnt++]=add;
        }
        for(int i=4;i<n;i++)
            for(int j=0;j<cnt;j++)
                if(F[j].ok && ptof(P[i],F[j])>eps){
                    dfs(i,j); break;
                }
        int tmp=cnt;
        cnt=0;
        for(int i=0;i<tmp;i++)
            if (F[i].ok) F[cnt++]=F[i];
}
double area(){
    double ret=0.0;
    for(int i=0; i<cnt;i++)
    ret+=area(P[F[i].a], P[F[i].b], P[F[i].c]);
    return ret/2.0;
}
double volume(){
    pt O(0,0,0);
    double ret=0.0;
    for(int i=0;i<cnt;i++)
        ret+=volume(O,P[F[i].a],P[F[i].b],P[F[i].c]);
    return fabs(ret/6.0);
}
int facetCnt_tri(){
    return cnt;
```

```
        }
        int facetCnt(){
                int ans=0;
                for(int i=0;i<cnt;i++){
                        bool nb=true;
                        for(int j=0;j<i;j++)
                                if (same(i,j)){
                                        nb=false; break;
                                }
                        if(nb) ans++;
                }
                return ans;
        }
};
_3DCH hull;
int main()
{
        while(~scanf("%d", &hull.n)){
        if(hull.n==0) break;
                for(int i=0;i<hull.n;i++)
                     scanf("%lf%lf%lf",&hull.P[i].x,&hull.P[i].y,&hull.P[i].z);
                hull.construct();
                printf("%0.2lf\n", hull.volume());
        }
        return 0;
}
---
///////******* Min-Cost Flow *******///////
#include <iostream>
#include <math.h>
#include <queue>
using namespace std;
int cap[202][202],pre[202],dis[202];       //node[0 = start, 1~n = male, n+1~2*n = female,
2*n+1 = end]
int cst[202][202];       //cost
bool h[202];
int n;
void init(){
      int i;
      memset(cap,0,sizeof(cap));
      memset(cst,0,sizeof(cst));
      for(i=1;i<=n;i++)
            cap[0][i]=cap[n+i][2*n+1]=1;
}
bool spfa(int s,int e){
      queue<int> q;
      int i,t;
      for(i=0;i<202;i++){
            dis[i]=2147483647;
            h[i]=false;
      }
      h[s]=true;
      pre[s]=s;   dis[s]=0;
      q.push(s);
      while(!q.empty()){
            t=q.front();      q.pop();
            h[t]=false;
            for(i=0;i<=(2*n+1);i++)
                  if(cap[t][i]>0&&dis[i]>dis[t]+cst[t][i]){
                        pre[i]=t;
                        dis[i]=dis[t]+cst[t][i];
                        if(!h[i]){
                              h[i]=true;
                              q.push(i);
                        }
                  }
      }
      return dis[e]<2147483647;
}
int mcf(int s,int e){
      int i,p;
      int mn=2147483647,cost=0;
      while(spfa(s,e)){
            p=e;
            while(pre[p]!=p){
                  mn=min(mn,cap[pre[p]][p]);
                  cost+=cst[pre[p]][p];
                  p=pre[p];
            }
            p=e;
            while(pre[p]!=p){
                  cap[pre[p]][p]-=mn;     cap[p][pre[p]]+=mn;
                  p=pre[p];
            }
      }
      return cost;
}
int main(){
      int i,j,a,b,c,d;
      while(true){
            cin>>n>>m;
            init();
            if(n==0)    break;
            for(i=0;i<m;i++){
                  cin>>a>>b>>c>>d;  //a to b, cap=c, cst=d
                  cap[a][b]+=c;
                  cst[a][b]+=c;     cst[a][b]-=c;
            }
```

```
            cout<<mcf(0,n-1)<<endl;
        }
        system("pause");
        return 0;
}
---
///////******* Min-Cost Flow fast *******/////// from www.csie.ntnu.edu.tw/~u91029/
typedef int Graph[10][10];  // adjacency matrix
Graph C, F, W;              // max cap,flow,cost
int p[10], d[10];           // min dis tree, min dis
int h[10];                  // modify weight
bool Dijkstra(int s, int t){
    /* init Dijkstra's Algorithm */
    memset(p, -1, sizeof(p));
    for (int i=0; i<10; ++i) d[i] = 1e9;
    /* min cost path. */
    d[s] = 0; p[s] = -s-1;
    for (int k=0; k<10; ++k){
        int a = -1, min = 1e9;
        for (int i=0; i<10; ++i)
            if (p[i] < 0 && d[i] < min)
                min = d[a = i];
        if (a == -1) break;
        p[a] = -p[a]-1;
        for (int i=a, j=0; j<10; ++j){
            if (p[j] >= 0) continue;
            int d1 = d[i] - (W[j][i] + h[j] - h[i]);
            if (F[j][i] > 0 && d1 < d[j])
                d[j] = d1, p[j] = -i-1;
            int d2 = d[i] + (W[i][j] + h[i] - h[j]);
            if (F[i][j] < C[i][j] && d2 < d[j])
                d[j] = d2, p[j] = -i-1;
        }
    }
    /* modify weight > 0,to use Dijkstra's Algorithm了F. */
    for (int i=0; i<10; ++i)
        if (h[i] < 1e9)    h[i] += d[i];
    return p[t] >= 0;
}
void MinCostFlow(int s, int t){
    memset(F, 0, sizeof(F));
    memset(h, 0, sizeof(h));
    int f = 0; c = 0;   // cap ,cost
    while (Dijkstra(s, t)){
        int df = 1e9, dc = 0;
        for (int j=t, i=p[t]; i!=j; i=p[j=i])
            df <?= (F[j][i] ? F[j][i] : C[i][j] - F[i][j]);
        for (int j=t, i=p[t]; i!=j; i=p[j=i])
            if (F[j][i])    F[j][i] -= df, dc -= W[j][i];
            else    F[i][j] += df, dc += W[i][j];
        f += df;
        c += df * dc;
    }
    cout << "max flow " << f ;
    cout << "min cost " << c;
}
---
///////******* Topological Sort *******///////
#include "iostream"
#include "list"
#include "algorithm"
using namespace std;
typedef struct node{
      int num;
      int endtime;
      bool operator<(node x){
            return endtime>x.endtime;
      }
};
node v[1000];
list<int>edge[1000];
bool visit[1000];
int n,m;    //n node, m edge
int time=0;
void init(){
      int i;
      for(i=0;i<n;i++){
            v[i].endtime=-1;
            v[i].num=i;
            visit[i]=false;
      }
}
void dfs(int x){
      visit[x]=true;
      list<int>::iterator i;
      for(i=edge[x].begin();i!=edge[x].end();i++)
            if(!visit[*i])    dfs(v[*i].num);
      v[x].endtime=time;
      time++;
}
int main(){
      int i;
      int a,b;
      cin>>n>>m;
      init();
      for(i=0;i<m;i++){
            cin>>a>>b;
            edge[a].push_back(b);
```

```
	}
	for(i=0;i<n;i++)
		if(!visit[i])	dfs(v[i].num);
	sort(v,v+n);
	for(i=0;i<n;i++)
		cout<<v[i].num<<endl;
	system("pause");
	return 0;
}
---
///////******* SCC *******///////
#include "iostream"
#include "algorithm"
#include "list"
using namespace std;
typedef struct node{
	int num;
	int endtime;
	bool operator<(node x){
		return endtime>x.endtime;
	}
};
node v[1000];
bool visit[1000];
int n,m;	//n node, m edge
list<int> fe[1000];
list<int> be[1000];
int time;
void dfsf(int x){
	visit[x]=true;
	list<int>::iterator i;
	for(i=fe[x].begin();i!=fe[x].end();i++)
		if(v[*i].endtime==-1&&!visit[*i])	dfsf(*i);
	v[x].endtime=time;
	time++;
}
void dfsb(int x){
	visit[x]=true;
	list<int>::iterator i;
	for(i=be[x].begin();i!=be[x].end();i++)
		if(!visit[*i])	dfsb(*i);
	cout<<x<<" ";
}
int main(){
	int i;
	int a,b;
	cin>>n>>m;
	for(i=0;i<n;i++){
		v[i].num=i;
		v[i].endtime=-1;
		visit[i]=false;
	}
	for(i=0;i<m;i++){
		cin>>a>>b;
		fe[a].push_back(b);
		be[b].push_back(a);
	}

	time=0;
	for(i=0;i<n;i++)
		if(v[i].endtime==-1)	dfsf(v[i].num);
	sort(v,v+n);
	for(i=0;i<n;i++)
		visit[i]=false;
	for(i=0;i<n;i++)
		if(!visit[v[i].num]){
			dfsb(v[i].num);
			cout<<endl;
		}
	system("pause");
	return 0;
}
---
///////******* RMQ *******////////
#include "iostream"
#include "math.h"
using namespace std;
int f[1000][1000];
int s[1000];
int n;
int dp(int x,int y){	//start = x, count = 2^y
	if(f[x][y]!=-1)	return f[x][y];
	if(y==0)	return s[x];
	else	return f[x][y]=min(dp(x,y-1),dp(x+(1<<(y-1)),y-1));
}
int rmq(int x,int y){	//start = x, end = y
	int i,j=0;
	for(i=1;i<(y-x+1);i*=2)
		j++;
	if(i>(x-y+1))	j--;
	return min(dp(x,j),dp(y-(1<<(j))+1,j));
}
void init(){
	int i,j;
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			f[i][j]=-1;
}
```

```
int main(){
	int i;
	int a,b;
	cin>>n;
	init();
	for(i=0;i<n;i++)
		cin>>s[i];
	while(true){
		cin>>a;
		if(cin.eof())	break;
		cin>>b;
		cout<<rmq(a,b)<<endl;
	}
	system("pause");
	return 0;
}
---
///////******* LCA *******///////
#include "iostream"
#include "list"
#include "math.h"
using namespace std;
int dep[1000];
int f[2000][2000];
int idx[1000];
int s[2000];
int p[2000];
int time;
int n;
int root=0;
list<int> edge[1000];
void init(){
	int i,j;
	for(i=0;i<n;i++){
		dep[i]=idx[i]=-1;
		time=0;
	}
	for(i=0;i<2*n;i++)
		for(j=0;j<2*n;j++)
			f[i][j]=-1;
	time=0;
}
void dfs(int x){
	if(idx[x]==-1)	idx[x]=time;
	s[time]=dep[x];	p[time]=x;	time++;
	list<int>::iterator i;
	for(i=edge[x].begin();i!=edge[x].end();i++)
		if(dep[*i]==-1){
			dep[*i]=dep[x]+1;
			dfs(*i);
			s[time]=dep[x];	p[time]=x;	time++;
		}
}
int dp(int x,int y){	//start = x, count = 2^y
	if(f[x][y]!=-1)	return f[x][y];
	if(y==0)	return x;
	int i,j;
	i=dp(x,y-1);	j=dp(x+(1<<(y-1)),y-1);
	if(s[i]>s[j])	return f[x][y]=j;
	else	return f[x][y]=i;
}
int rmq(int x,int y){	//start = x, end = y
	int i,j=0;
	for(i=1;i<(y-x+1);i*=2)
		j++;
	if(i>(y-x+1))	j--;
	int a,b;
	a=dp(x,j);	b=dp(y-(1<<(j))+1,j);
	if(s[a]>s[b])	return b;
	else	return a;
}
int lca(int x,int y){	//node x,y
	return p[rmq(min(idx[x],idx[y]),max(idx[x],idx[y]))];
}
int main(){
	int i,j;
	int a,b;
	cin>>n;
	init();
	for(i=0;i<n-1;i++){
		cin>>a;
		if(cin.eof())	break;
		cin>>b;
		edge[a].push_back(b);
		edge[b].push_back(a);
	}
	dep[root]=0;
	dfs(root);
	while(true){
		cin>>a;
		if(cin.eof())	break;
		cin>>b;
		cout<<lca(a,b)<<endl;
	}
	system("pause");
	return 0;
}
---
```

```
///////******* Bridge *******///////
#include "iostream"
#include "list"
#include "math.h"
using namespace std;
bool dfsedge[1000][1000];
bool bridge[1000][1000];
int dep[1000];
int low[1000];
list<int> edge[1000];
int n,m;    //n node, m edge
int root;
void init(){
      int i,j;
      for(i=0;i<n;i++){
            dep[i]=low[i]=-1;
            for(j=0;j<n;j++)
                  dfsedge[i][j]=bridge[i][j]=false;
      }
}
void dfs(int x){
      list<int>::iterator i;
      for(i=edge[x].begin();i!=edge[x].end();i++)
            if(dep[*i]==-1){
                  dep[*i]=dep[x]+1;
                  dfsedge[x][*i]=dfsedge[*i][x]=true;
                  dfs(*i);
            }
}
int findlow(int x){
      if(low[x]!=-1)    return low[x];
      low[x]=dep[x];
      list<int>::iterator i;
      for(i=edge[x].begin();i!=edge[x].end();i++){
            if(dfsedge[x][*i]){
                  if(dep[*i]>dep[x])      low[x]=min(low[x],findlow(*i));
            }
            else  low[x]=min(low[x],dep[*i]);
      }
      return low[x];
}
bool chkbridge(int x,int y){
      return findlow(y)>dep[x];
}
int main(){
      int i;
      int a,b;
      list<int>::iterator j;
      cin>>n>>m;
      for(i=0;i<m;i++){
            cin>>a>>b;
            edge[a].push_back(b);
            edge[b].push_back(a);
      }
      init();
      dep[root]=0;
      dfs(root);
      for(i=0;i<n;i++)
            for(j=edge[i].begin();j!=edge[i].end();j++){
                  bridge[i][*j]=chkbridge(i,*j);
                  if(bridge[i][*j]) cout<<i<<" "<<*j<<endl;
            }
      system("pause");
      return 0;
}
---
///////******* BCC *******///////
#include "iostream"
#include "list"
#include "math.h"
using namespace std;
bool bridge[1000][1000];
int dep[1000];
int low[1000];
bool visit[1000];
bool dfsedge[1000][1000];
int root;
int n,m;    //n node,m edge
list<int>edge[1000];
void init(){
      int i,j;
      for(i=0;i<n;i++){
            dep[i]=low[i]=-1;
            visit[i]=false;
            for(j=0;j<n;j++)
                  bridge[i][j]=dfsedge[i][j]=false;
      }
}
void dfs(int x){
      list<int>::iterator i;
      for(i=edge[x].begin();i!=edge[x].end();i++){
            if(dep[*i]==-1){
                  dep[*i]=dep[x]+1;
                  dfsedge[x][*i]=dfsedge[*i][x]=true;
                  dfs(*i);
            }
      }
}
```

```
int findlow(int x){
	if(low[x]!=-1)	return low[x];
	low[x]=dep[x];
	list<int>::iterator i;
	for(i=edge[x].begin();i!=edge[x].end();i++){
		if(dfsedge[x][*i]){
			if(dep[*i]>dep[x])	low[x]=min(low[x],findlow(*i));
		}
		else  low[x]=min(low[x],dep[*i]);
	}
	return low[x];
}
void printbcc(int x){
	list<int>::iterator i;
	visit[x]=true;
	for(i=edge[x].begin();i!=edge[x].end();i++)
		if(!bridge[x][*i]&&!visit[*i]){
			printbcc(*i);
		}
	cout<<x<<" ";
}
int main(){
	int i;
	int a,b;
	cin>>n>>m;
	init();
	for(i=0;i<m;i++){
		cin>>a>>b;
		edge[a].push_back(b);
		edge[b].push_back(a);
	}
	dep[root]=0;
	dfs(root);
	list<int>::iterator j;
	for(i=0;i<n;i++)
		for(j=edge[i].begin();j!=edge[i].end();j++){
			if(findlow(*j)>dep[i])  bridge[i][*j]=bridge[*j][i]=true;
		}
	for(i=0;i<n;i++)
		if(!visit[i]){
			printbcc(i);
			cout<<endl;
		}
	system("pause");
	return 0;
}
---
///////******* Articulation Point *******///////
#include "iostream"
#include "list"
#include "math.h"
using namespace std;
bool ap[1000]={false};
int edge[1000][1000];
int dep[1000];
int low[1000];
int n,m;	//n vertex, m edge
int root=0;
void init(){
	int i,j;
	for(i=0;i<n;i++){
		dep[i]=low[i]=-1;
		for(j=0;j<n;j++)
			edge[i][j]=0;
	}
}
void dfs(int x){
	int i;
	for(i=0;i<n;i++)
		if(edge[x][i]==1&&dep[i]==-1){
			dep[i]=dep[x]+1;
			edge[x][i]=edge[i][x]=2;
			dfs(i);
		}
}
int findlow(int x){
	if(low[x]!=-1)	return low[x];
	else{
		int i;
		low[x]=dep[x];
		for(i=0;i<n;i++){
			if(edge[x][i]==2&&dep[x]<dep[i])	low[x]=min(low[x],findlow(i));
			else if(edge[x][i]==1)  low[x]=min(low[x],dep[i]);
		}
		return low[x];
	}
}
bool checkap(int x){
	int i;
	if(x==root){
		int count=0;
		for(i=0;i<n;i++)
			if(edge[x][i]==2) count++;
		if(count>=2)	return true;
		else  return false;
	}
	else{
		for(i=0;i<n;i++)
```

```
                if(edge[x][i]==2 &&dep[i]>dep[x]){
                        if(findlow(i)>=dep[x])  return true;
                }
            return false;
      }
}
int main(){
      int i;
      int a,b;
      cin>>n>>m;
      init();
      for(i=0;i<m;i++){
            cin>>a>>b;
            edge[a][b]=edge[b][a]=1;
      }
      dep[root]=0;
      dfs(root);
      for(i=0;i<n;i++){
            if(checkap(i))    ap[i]=true;
            else  ap[i]=false;
      }
      for(i=0;i<n;i++)
            if(ap[i])   cout<<i<<endl;
      system("pause");
      return 0;
}
---
///////******* Prim *******///////
#include "iostream"
using namespace std;
int map[10001][10001];
int dis[10001];
bool add[10001];
int ans;
int n, m;
int searchmin(){
      int i,t = 999999999, = 0;
      for(i= 1 ; i<= n ; i+)
            if(!add[i && dis[i]< t){
                  t = dis[i;
                  s = i
            }
      ans += dis[s];
      add[s] = true;
      return s;
}
void update(int x){
      int i
      for(i= 1 ; i<= n ; i+)
            if(dis[i > map[x][i)    dis[i = map[x][i;
}
int main(array<System::String ^> ^args){
      int i,j;
      int ts, te, td;
      while(true){
            cin >> n >> m;
            if(n != 0 && m != 0){
                  ans = 0;
                  for(i= 1 ; i<= n ; i++){
                        add[i] = false;
                        dis[i] = 999999999;
                        for(j = 1 ; j <= n ; j++)
                              map[i][j] = 999999999;
                  }
                  dis[1] = 0;
                  for(i= 0 ; i< m ; i++){
                        cin >> ts >> te >> td;
                        if(td<map[ts][te]){
                              map[ts][te] = td;
                              map[te][ts] = td;
                        }
                  }
                  for(i= 0 ; i< n ;i++){
                        ts = searchmin();
                        if(ts > 0)  update(ts);
                        else{
                              cout << "-1" << endl;
                              goto out;
                        }
                  }
                  cout << ans << endl;
out:;
            }
            else  break;
      }
    system("pause");
    return 0;
}

---
///////******* Kruskal *******///////
#include "iostream"
#include "algorithm"
using namespace std;
typedef struct edge{
      int s,e,d;
      edge(){
            s = e = d = 0;
```

```
	}
	bool operator <(edge t)const{
		return d < t.d;
	}
};
edge e[1000000];
int pre[10001];
int root(int s){
	if(pre[s] == s)	return s;
	else	return root(pre[s]);
}
bool chkset(int a, int b){
	return root(a) != root(b))
}
void join(int a, int b){
	pre[root(a)] = root(b);
}
int main(array<System::String ^> ^args){
	ios_base::sync_with_stdio(false);
	int n, m,sum,ce;
	int i;
	while(true){
		cin >> n >> m;
		if(n == 0 && m == 0)	break;
		else{
			for(c = 0 ; c < m ; c++)
				cin >> e[c].s >> e[c].e >> e[c].d;
			sort(e, e + m);
			for(i = 1 ; i <= n ; i++)
				pre[i] = i;
			sum = 0;	ce = 0;
			for(i = 0 ; i < m ; i ++){
				if(chkset(e[i].s, e[i].e)){
					sum += e[i].d;
					join(e[i].s, e[i].e);
					ce++;
				}
			}
			if(ce == (n - 1)) cout << sum << endl;
			else	cout << "-1" << endl;
		}
	}
    system("pause");
    return 0;
}
---
//////******* Minimum Edge-disjoint Path Cover *******////// from
www.csie.ntnu.edu.tw/~u91029/
int adj[9][9];	// adjacency matrix
int in[9], out[9];	// degrees
deque<int> path;	// Euler Path
deque<int> pos;	// position in Euler Path
int Find_Start_Vertex(){
    for (int i=0; i<10; ++i)
        if (out[i] > in[i])
            return i;
    for (int i=0; i<10; ++i)
        if (out[i] > 0 && out[i] == in[i])
            return i;
    return -1;
}
void EulerPath(int x){
    if (out[x] > 0){
        // find Euler Path.
        for (int y=0; y<9; ++y)
            if (adj[x][y]){
                adj[x][y]--; out[x]--; in[y]--;
                EulerPath(y);
            }
    }
    else{
        for (int y=0; y<9; ++y)
            if (out[y] > in[y]){
                EulerPath(y);
                break;
            }
        pos.push_front(path.size());	// record position inserted
    }
    path.push_front(x);
}
void Minimum_Edge_Disjoint_Path_Cover(){
    memset(in, 0, sizeof(in));
    memset(out, 0, sizeof(out));
    for (int x=0; x<9; ++x)
        for (int y=0; y<9; ++y){
            out[x] += adj[x][y];
            in[y] -= adj[x][y];
        }
    int s = FindStartVertex();
    if (s == -1) return;
    path.clear();
    pos.clear();
    EulerCicuit(s);
    cout << "numbers of path " << pos.size();
}
---
//////******* RB_Tree *******//////
#include<iostream>
```

```
template<class T>
class rb_tree{
private:
struct node{
	node *l,*r,*p;	//left,right,parent
	T k;	//key
	bool clr;	//false = red, true = black
	int cnt;	//numbers of nodes below
	void operator=(node tmp){	//Note : don't copy cnt
		l=tmp.l;	r=tmp.r;	p=tmp.p;
		k=tmp.k;	clr=tmp.clr;
	}
	node(){
		cnt=1;	clr=false;
	}
};
node *root,*nil;
void LEFT_ROTATE(node* x){
	node* y=x->r;
	x->r=y->l;
	if(y->l!=nil)	y->l->p=x;
	y->p=x->p;
	if(x->p==nil)	root=y;
	else if(x==x->p->l)	x->p->l=y;
	else	x->p->r=y;
	y->l=x;	x->p=y;
	x->cnt=x->l->cnt+x->r->cnt+1;
	y->cnt=y->l->cnt+y->r->cnt+1;
}
void RIGHT_ROTATE(node* x){
	node* y=x->l;
	x->l=y->r;
	if(y->r!=nil)	y->r->p=x;
	y->p=x->p;
	if(x->p==nil)	root=y;
	else if(x==x->p->l)	x->p->l=y;
	else	x->p->r=y;
	y->r=x;	x->p=y;
	x->cnt=x->l->cnt+x->r->cnt+1;
	y->cnt=y->l->cnt+y->r->cnt+1;
}
node* TREE_MIN(node* x){
	while(x->l!=nil)	x=x->l;
	return x;
}
node* TREE_MAX(node* x){
	while(x->r!=nil)	x=x->r;
	return x;
}
```

```
node* TREE_SUCCESSOR(node* x){
	if(x->r!=nil)	return TREE_MIN(x->r);
	node* y=x->p;
	while(y!=nil&&x==y->r){
		x=y;	y=y->p;
	}
	return y;
}
node* TREE_PREDECESSOR(node* x){
	if(x->l!=nil)	return TREE_MAX(x->l);
	node* y=x->p;
	while(y!=nil&&x==y->l){
		x=y;	y=y->p;
	}
	return y;
}
void RB_INSERT_FIX(node* z){
	node* y;
	while(z->p->clr==false){
		if(z->p==z->p->p->l){
			y=z->p->p->r;
			if(y->clr==false){
				z->p->clr=y->clr=true;
				z->p->p->clr=false;
				z=z->p->p;
			}
			else{
				if(z==z->p->r){
					z=z->p;
					LEFT_ROTATE(z);
				}
				z->p->clr=true;
				z->p->p->clr=false;
				RIGHT_ROTATE(z->p->p);
			}
		}
		else{
			y=z->p->p->l;
			if(y->clr==false){
				z->p->clr=y->clr=true;
				z->p->p->clr=false;
				z=z->p->p;
			}
			else{
				if(z==z->p->l){
					z=z->p;
					RIGHT_ROTATE(z);
				}
				z->p->clr=true;
```

```
                        z->p->p->clr=false;
                        LEFT_ROTATE(z->p->p);
                    }
                }
            }
            root->clr=true;
}
void RB_DELETE_FIX(node* x){
            node *w;
            while(x!=root&&x->clr){
                  if(x==x->p->l){
                        w=x->p->r;
                        if(!w->clr){
                              w->clr=true;      x->p->clr=false;
                              LEFT_ROTATE(x->p);
                              w=x->p->r;
                        }
                        if(w->l->clr&&w->r->clr){
                              w->clr=false;
                              x=x->p;
                        }
                        else{
                              if(w->r->clr){
                                    w->l->clr=true;   w->clr=false;
                                    RIGHT_ROTATE(w);
                                    w=x->p->r;
                              }
                              w->clr=x->p->clr;
                              x->p->clr=w->r->clr=true;
                              LEFT_ROTATE(x->p);
                              x=root;
                        }
                  }
                  else{
                        w=x->p->l;
                        if(!w->clr){
                              w->clr=true;      x->p->clr=false;
                              RIGHT_ROTATE(x->p);
                              w=x->p->l;
                        }
                        if(w->l->clr&&w->r->clr){
                              w->clr=false;
                              x=x->p;
                        }
                        else{
                              if(w->l->clr){
                                    w->r->clr=true;   w->clr=false;
                                    LEFT_ROTATE(w);
                                    w=x->p->l;
                              }
                              w->clr=x->p->clr;
                              x->p->clr=w->l->clr=true;
                              RIGHT_ROTATE(x->p);
                              x=root;
                        }
                  }
            }
            x->clr=true;
}
void RB_DELETE(node* z){
            node *x,*y;
            if(z->l==nil||z->r==nil)      y=z;
            else  y=TREE_SUCCESSOR(z);
            x=y->p;
            while(x!=nil){
                  x->cnt--;   x=x->p;
            }
            if(y->l!=nil)     x=y->l;
            else  x=y->r;
            x->p=y->p;
            if(y->p==nil)     root=x;
            else if(y==y->p->l)     y->p->l=x;
            else  y->p->r=x;
            if(y!=z)    z->k=y->k;
            if(y->clr==true)  RB_DELETE_FIX(x);
            delete y;
}
void RB_INSERT(node* z){
            node *y=root,*x=nil;
            while(y!=nil){
                  y->cnt++;
                  x=y;
                  if(z->k<y->k)     y=y->l;
                  else  y=y->r;
            }
            z->p=x;
            if(x==nil)  root=z;
            else if(z->k<x->k)      x->l=z;
            else  x->r=z;
            z->l=z->r=nil;
            z->clr=false;
            RB_INSERT_FIX(z);
}
node* RB_FIND(int x){
            node *y=root;
            while(y->k!=x){
                  if(y->k<x)  y=y->r;
                  else  y=y->l;
```

```
        }
        return y;
    }
    node* RB_NTH(int x){
        node *y=root;
        int u;
        while(true){
            if(y->l!=nil)     u=y->l->cnt;
            else u=0;
            if(u>=x)    y=y->l;
            else if(u==x-1)   return y;
            else{
                y=y->r;
                x-=u+1;
            }
        }
    }
    void RB_DEL(node* x){
        if(x->l!=nil)     RB_DEL(x->l);
        if(x->r!=nil)     RB_DEL(x->r);
        if(x==root) x=root=nil;
        else if(x!=nil)         delete x;
    }
    public:
        struct iterator{
            private:
                rb_tree<T> *tmp;
            public:
            node* x;
            iterator(){}
            iterator(node* u,rb_tree<T> *v){
                x=u;  tmp=v;
            }
            T operator *(){
                return      x->k;
            }
            bool operator==(iterator tmp){
                return x==tmp.x;
            }
            bool operator!=(iterator tmp){
                return x!=tmp.x;
            }
            iterator operator++(int){     //O(lg n)
                iterator y=iterator(x,tmp);
                x=tmp->TREE_SUCCESSOR(x);
                return y;
            }
            iterator operator++(){  //O(lg n)
                x=tmp->TREE_SUCCESSOR(x);
                return iterator(x,tmp);
            }
            iterator operator--(int){     //O(lg n)
                iterator y=iterator(x,tmp);
                x=tmp->TREE_PREDECESSOR(x);
                return y;
            }
            iterator operator--(){  //O(lg n)
                x=tmp->TREE_PREDECESSOR(x);
                return iterator(x,tmp);
            }
    };
    T operator[](int y){
        return RB_NTH(y+1)->k;
    }
    rb_tree(){
        nil=new node();
        nil->cnt=0; nil->clr=true;
        root=nil;
    }
    void insert(T x){ //O(lg n)
        node *y=new node();
        y->k=x;
        RB_INSERT(y);
    }
    void erase(T x){  //O(lg n)
        node *y=new node();
        y->k=x;
        RB_INSERT();
    }
    void erase(iterator x){ //O(lg n)
        RB_DELETE(x.x);
    }
    iterator find(int x){   //O(lg n)
        return iterator(RB_FIND(x),this);
    }
    iterator nth_element(int x){  //from 1 to n, O(lg n)
        return iterator(RB_NTH(x),this);
    }
    iterator end(){   //O(lg n)
        return iterator(nil,this);
    }
    iterator begin(){ //O(lg n)
        return iterator(TREE_MIN(root),this);
    }
    int size(){
        return root->cnt;
    }
    void clear(){     //O(n)
```

```
            if(size()>0)      RB_DEL(root);
      }
};
int main(){
      rb_tree<int> t;
      rb_tree<int>::iterator i;
      system("pause");
      return 0;
}
---
///////******* Binary Indexed Tree + Binary Search *******///////
/// ACM 11522 - Permutation
Binary Indexed Tree + Binary Search
#include<iostream>
#include<memory.h>
using namespace std;
int m[50010],k;
int read(int i){
        int sum=0;
        while(i>0){
                sum+=m[i];
                i-=i&-i;
        }
        return sum;
}
int update(int i,int val){
        while(i<=k){
                m[i]+=val;
                i+=i&-i;
        }
}
int bi(int val){
        int r=k,l=1,tmp,v1,v2;
        while(1){
                tmp=(r+l)/2;
                v1=read(tmp);
                v2=read(tmp-1);
                if(v1==val && v1!=v2)           return tmp;
                if(v1>val||(v1==val&&v1==v2))   r=tmp-1;
                else                            l=tmp+1;
        }
}
int main(){
        int T,tmp,p;
        scanf("%d",&T);
        while(T--){
                scanf("%d",&k);
                memset(m,0,(k+1)*sizeof(int));
                for(int i=1;i<=k;++i)   update(i,1);
                for(int i=1;i<=k;++i){
                        scanf("%d",&tmp);
                        if(i!=1)        printf(" ");
                        p=bi(tmp+1);
                        printf("%d",p);
                        update(p,-1);
                }
                printf("\n");
        }
        return 0;
}
---
///////******* KMP *******///////
#include "iostream"
#include "string"
using namespace std;
string s,t;
int pi[1000];
void cpf(){
      int i;
      int k=0;
      pi[0]=0;
      for(i=1;i<t.length();i++){
            while(k>0&&t[k]!=t[i])  k=pi[k];
            if(t[k]==t[i])    k++;
            pi[i]=k;
      }
}
void kmp(){
      int i;
      int k=0;
      for(i=0;i<s.length();i++){
            while(k>0&&t[k]!=s[i])  k=pi[k];
            if(t[k]==s[i])    k++;
            if(k==t.length()){
                  cout<<"mach at : "<<(i-t.length()+1)<<endl;
                  k=pi[k];
            }
      }
}
int main(){
      cin>>s>>t;
      cpf();
      kmp();
      system("pause");
      return 0;
}
---
///////******* Multi-matching *******///////  from www.csie.ntnu.edu.tw/~u91029/
```

```
int N = 1000;           // numbers of p
char T[1000+1];
char P[1000][1000+1];
bool occur[1000+1];     // record if p apear in t
int equiv[1000+1];      // record min of index
struct TrieNode{
     TrieNode* l[26], *failure, *suffix;
     int index;  // idx of p
 } *root;
void init(){
     memset(trieNode, 0, sizeof(trieNode));
     memset(occur, 0, sizeof(occur));
     memset(equiv, 0, sizeof(equiv));
     root = new TrieNode();
}
void free(TrieNode* p = root){// del trie
    for (int i=0; i<26; ++i)
        if (p->l[i])    free(p->i[i]);
    delete p;
}
void add(char* s, int index){// insert trie
    TrieNode* p = root;
    for (; *s; ++s){
        if (!p->l[*s - 'A'])  p->l[*s - 'A'] = new TrieNode();
        p = p->l[*s - 'A'];
    }
    // if appeared,// record min indexed string,// or add to trie.
    if (p->index) equiv[index] = p->index;
    else    p->index = index;
}
// failure link,suffix link // by BFS // O(sum(Pi))
void build(){
    TrieNode* q[100000], **qf = q, **qb = q;
    *qb++ = root;
    while (qf < qb){
        TrieNode* p = *qf++;
        for (int i=0; i<26; ++i)
            if (p->l[i]){
                TrieNode* q = p->failure;
                while (q && !q->l[i]) q = q->failure;
                if (q){
                    p->l[i]->failure = q->l[i];
                    p->l[i]->suffix = (q->l[i]->index ?
                        q->l[i] : q->l[i]->suffix);
                }
                else    p->l[i]->failure = p->l[i]->suffix = root;
                *qb++ = p->l[i];
            }
    }
}
// comopare T by trie // O(T+K)
void match(char* s){
    TrieNode* p = root;
    for (; *s; ++s){
        // try current char,// failure link if fail // sum O(T).
        while (p && !p->l[*s - 'A']) p = p->failure;
        if (p) p = p->l[*s - 'A'];
        else p = root;
        // match ! // suffix link for all suffix. // sum O(K).
        if (p->index)
            for (TrieNode* q = p; q; q = q->suffix)
                occur[q->index] = true;
    }
}
int main(){
    init();
    for (int i=0; i<N; ++i)
      add(P[i], i+1);
    build();    // failure link,suffix link
    match(T);
    free();
    for (int i=1; i<=N; ++i)
        // success match
        if (occur[i] || occur[equiv[i]])    cout << i << "matched";
        else    cout << i << "unmatched";
    return 0;
}
---
///////******* LIS *******///////
#include "iostream"
using namespace std;
int f[100010],b[100010];
int bsearch(int x, int end){
      int s, e, p;
      s = -1;     e = end;
      if(e == -1 || b[e] < x) return ++e;
      while(s != (e - 1)){
            p = (s + e) / 2;
            if(b[p] < x)      s = p;
            else  e = p;
      }
      return e;
}
int main(){
      int n,idx,lis = 0;
      int i;
      scanf("%d",&n);
      for(i = 0 ; i < n ; i++){
```

```
            scanf("%d", &f[i]);
            b[i] = 0;
      }
      for(i = 0 ; i < n ; i++){
            idx = bsearch(f[i], lis - 1);
            b[idx] = f[i];
            if(idx == lis)    lis++;
      }
      printf("%d", lis);
    system("pause");
    return 0;
}
---
///////******* LIS - STL *******/////// from www.csie.ntnu.edu.tw/~u91029/
int LIS(vector<int>& s){     // sequence from s
    if (s.size() == 0) return 0;    // special case
    vector<int> v;
    v.push_back(s[0]);  // prevent v.back() RE
    for (int i = 1; i < s.size(); ++i){
        int n = s[i];
        if (n > v.back())   v.push_back(n);
        else    *lower_bound(v.begin(), v.end(), n) = n;
    }
    return v.size();
}
---
///////******* Joseph's Problem *******///////
//NTUJudge 1066 - Jump
#include<stdio.h>
int main(){
    int T,n,k,ans[3],i,tmp;
    scanf("%d",&T);
    while(T--){
        scanf("%d %d",&n,&k);
        for(i=0;i<3;++i)    ans[i]=(k+i)%(i+1);
        for(i=2;i<=n;++i)   ans[0]=(k%i+ans[0])%i;
        for(i=3;i<=n;++i)   ans[1]=(k%i+ans[1])%i;
        for(i=4;i<=n;++i)   ans[2]=(k%i+ans[2])%i;
        printf("%d %d %d\n",ans[2]+1,ans[1]+1,ans[0]+1);
    }
    return 0;
}
---
///////******* Big Num *******///////
#include "iostream"
#include "string"
#include "math.h"
using namespace std;
typedef struct BigNumber{
      int *Number;
      int Length;
      int SigneNumber;
      BigNumber(string number){
            int i;
            if(number[0] == '-'){
                  Length = (number.length() - 1);
                  Number = new int[Length];
                  SigneNumber = -1;
                  for(i=1;i<Length+1;i++)
                        Number[i - 1] = (number[i] - '0');
            }
            else{
                  Length = number.length();
                  Number = new int[Length];
                  SigneNumber = 1;
                  for(i = 0 ; i < Length ; i++)
                        Number[i] = (number[i] - '0');
            }
      }
      BigNumber(int number[],int length,int signenumber){
            int i;
            Length = length;
            Number = new int[Length];
            SigneNumber = signenumber;
            for(i = 0 ; i < Length ; i++){
                  Number[i] = number[i];
            }
      }
      BigNumber(){}
      BigNumber operator+(BigNumber AfterNumber){
            int SumLength;    //    length of output Number
            int OutLength;
            int j;
            if((SigneNumber > 0) && (AfterNumber.SigneNumber > 0)){
                  SumLength = max(Length, AfterNumber.Length) + 1;
                  int *Sum = new int[SumLength];
                  int a, b;
                  a = SumLength - Length;
                  b = SumLength - AfterNumber.Length;
                  for(j = 0 ; j < SumLength ; j++)
                        Sum[j] = 0;
                  for(j = (SumLength - 1) ; j > 0 ; j--){
                        if(j >= a)  Sum[j] += Number[j - a];
                        if(j >= b)  Sum[j] += AfterNumber.Number[j - b];
                        if(Sum[j] > 9){
                              Sum[j - 1] += Sum[j] / 10;
                              Sum[j] %= 10;
                        }
```

```
            }
            int d;
            for(j = 0 ; j < SumLength ; j++)
                  if(Sum[j] != 0){
                        d = j;
                        break;
                  }
            OutLength = SumLength - d;
            int *Out = new int[OutLength];
            for(j = 0 ; j < OutLength ; j++)
                  Out[j] = Sum[j + d];
            return BigNumber(Out, OutLength, 1);
      }
      else if((SigneNumber < 0) && (AfterNumber.SigneNumber > 0)){
            BigNumber Temp;
            Temp.Length = Length;
            Temp.Number = Number;
            Temp.SigneNumber = 1;
            return AfterNumber - Temp;
      }
      else if((SigneNumber > 0) && (AfterNumber.SigneNumber < 0)){
            BigNumber Temp1;
            Temp1.Length = AfterNumber.Length;
            Temp1.Number = AfterNumber.Number;
            Temp1.SigneNumber = 1;
            BigNumber Temp2;
            Temp2.Length = Length;  Temp2.Number = Number;
            Temp2.SigneNumber = 1;
            return Temp2 - Temp1;
      }
      else{
            BigNumber Temp1;
            Temp1.Length = AfterNumber.Length;
            Temp1.Number = AfterNumber.Number;
            Temp1.SigneNumber = 1;
            BigNumber Temp2;
            Temp2.Length = Length;
            Temp2.Number = Number;
            Temp2.SigneNumber = 1;
            BigNumber Ans;
            Ans = Temp1 + Temp2;
            return BigNumber(Ans.Number, Ans.Length, -1);
      }
}
BigNumber operator-(BigNumber AfterNumber){
      int DisLength;
      int OutLength;    //    length of output Number
      int Sign = 1;
      int i;
      if((SigneNumber > 0) && (AfterNumber.SigneNumber > 0)){
            BigNumber Temp1;
            Temp1.Length = Length;
            Temp1.Number = Number;
            Temp1.SigneNumber = SigneNumber;
            BigNumber Temp2;
            Temp2.Length = AfterNumber.Length;
            Temp2.Number = AfterNumber.Number;
            Temp2.SigneNumber = AfterNumber.SigneNumber;
            if(Temp1 == Temp2){
                  int *Out = new int[0];
                  OutLength = 0;
                  return BigNumber(Out, OutLength, Sign);
            }
            if(Temp2 > Temp1){
                  BigNumber Tmp;
                  Tmp = Temp2;
                  Temp2  = Temp1;
                  Temp1 = Tmp;
                  Sign = -1;
            }
            DisLength = Temp1.Length;
            int *Dis = new int[DisLength];
            int a, b;
            a = DisLength - Temp2.Length;
            for(i = (DisLength - 1) ; i > -1 ; i--)
                  Dis[i] = 0;
            for(i = (DisLength - 1) ; i > -1 ; i--){
                  Dis[i] += Temp1.Number[i];
                  if(i >= (a))      Dis[i] -= Temp2.Number[i - a];

                  if(Dis[i] < 0){
                        Dis[i - 1]--;
                        Dis[i] += 10;
                  }
            }
            for(i = 0 ; i < DisLength ; i++)
                  if(Dis[i] != 0){
                        b = i;
                        break;
                  }
            OutLength = DisLength - b;
            int *Out = new int[OutLength];
            for(i = 0 ; i < OutLength ; i++)
                  Out[i] = Dis[i + b];
            delete Dis;
            return BigNumber(Out, OutLength, Sign);
      }
      else if((SigneNumber < 0) && (AfterNumber.SigneNumber > 0)){
```

```
            BigNumber Temp1;
            Temp1.Length = AfterNumber.Length;
            Temp1.Number = AfterNumber.Number;
            Temp1.SigneNumber = 1;
            BigNumber Temp2;
            Temp2.Length = Length;
            Temp2.Number = Number;
            Temp2.SigneNumber = 1;
            BigNumber Ans;
            Ans = Temp1 + Temp2;
            return BigNumber(Ans.Number, Ans.Length, -1);
      }
      else if((SigneNumber > 0) && (AfterNumber.SigneNumber < 0)){
            BigNumber Temp1;
            Temp1.Length = AfterNumber.Length;
            Temp1.Number = AfterNumber.Number;
            Temp1.SigneNumber = 1;
            BigNumber Temp2;
            Temp2.Length = Length;
            Temp2.Number = Number;
            Temp2.SigneNumber = 1;
            return Temp1 + Temp2;
      }
      else{
            BigNumber Temp1;
            Temp1.Length = AfterNumber.Length;
            Temp1.Number = AfterNumber.Number;
            Temp1.SigneNumber = 1;
            BigNumber Temp2;
            Temp2.Length = Length;
            Temp2.Number = Number;
            Temp2.SigneNumber = 1;
            return Temp1 - Temp2;
      }
}
BigNumber operator*(BigNumber AfterNumber){
      int MutLength;
      int OutLength;
      int Sign;
      int j, j, P;
      if(SigneNumber == AfterNumber.SigneNumber)      Sign = 1;
      else  Sign = -1;
      MutLength = Length + AfterNumber.Length;
      int *Mut = new int[MutLength];
      for(j = 0 ; j < MutLength ; j++)
            Mut[j] = 0;
      for(j = (AfterNumber.Length - 1) ; j > -1  ; j--)
            for(j = (Length - 1) ; j > -1  ; j--){
                  P = j + j + 1;
                  Mut[P] += Number[j] * AfterNumber.Number[j];
                  if(Mut[P] > 9){
                        Mut[P - 1] += Mut[P] / 10;
                        Mut[P] %= 10;
                  }
            }
      int a;
  a = MutLength;
      for(j = 0 ; j < MutLength ; j++)
            if(Mut[j] != 0)
            {
                  a = j;

                  break;
            }
      OutLength = MutLength - a;
      int *Out = new int[OutLength];
      for(j = 0 ; j < (OutLength) ; j++)
            Out[j] = Mut[j + a];
      delete Mut;
      return BigNumber(Out, OutLength, Sign);
}
BigNumber operator/(BigNumber AfterNumber){
      int DivLength;
      int OutLength;
      int Sign;
      int j, j, P;
      BigNumber Temp1, Temp2;
      BigNumber Temp3, Temp4;
      Temp1.Length = Length;
      Temp1.Number = Number;
      Temp1.SigneNumber = 1;
      Temp2 = AfterNumber;
      Temp2.SigneNumber = 1;
      DivLength = abs(Temp1.Length - Temp2.Length) + 1;
      int *Div = new int[DivLength];
      for(j = 0 ; j < DivLength ; j++)
            Div[j] = 0;
      for(j = 0 ; j < DivLength ; j++){
            P = DivLength - j - 1;
            int *tmp = new int[Temp2.Length + P];
            for(j = 0 ; j < Temp2.Length ; j++)
                  tmp[j] = Temp2.Number[j];
            for(j = 0 ; j < P ; j++)
                  tmp[j + Temp2.Length] = 0;
            Temp3 = BigNumber(tmp, (Temp2.Length + P), Temp2.SigneNumber);
            while(true){
                  if(Temp1 >= Temp3){
                        Temp4 = Temp1 - Temp3;
```

```
                        Temp1 = Temp4;
                        Div[j]++;
                    }
                    else  break;
                }
                delete tmp;
            }
            if(SigneNumber == AfterNumber.SigneNumber)      Sign = 1;
            else  Sign = -1;
            int a;
            a = DivLength;
            for(j = 0 ; j < DivLength ; j++)
                if(Div[j] != 0){
                    a = j;
                    break;
                }
            OutLength = DivLength - a;
            int *Out = new int[OutLength];
            for(j = 0 ; j < OutLength ; j++)
                Out[j] = Div[j + a];
            delete Div;
            return BigNumber(Out, OutLength, Sign);
}
BigNumber operator%(BigNumber AfterNumber){
            BigNumber Temp;
            Temp.Length = Length;
            Temp.Number = Number;
            Temp.SigneNumber = SigneNumber;
            return Temp - (AfterNumber * (Temp / AfterNumber));
}
BigNumber operator^(int x){
            BigNumber Temp1;
            int j;
            Temp1.Length = Length;
            Temp1.Number = Number;
            Temp1.SigneNumber = SigneNumber;
            for(j = 1 ; j < x ; j++)
                Temp1 = Temp1 * Temp1;
            return Temp1;
}
void operator=(BigNumber AfterNumber){
            Number = AfterNumber.Number;
            Length = AfterNumber.Length;
            SigneNumber = AfterNumber.SigneNumber;
}
bool operator==(BigNumber AfterNumber){
            int j;
            if(Length == AfterNumber.Length){
                if(SigneNumber == AfterNumber.SigneNumber){
                    for(j = 0 ; j < Length ; j++)
                        if(Number[j] != AfterNumber.Number[j])    return false;
                    return true;
                }
                else  return false;
            }
            else  return false;
}
bool operator!=(BigNumber AfterNumber){
            BigNumber Temp;
            Temp.Length = Length;
            Temp.Number = Number;
            Temp.SigneNumber = SigneNumber;
            if(Temp == AfterNumber) return false;
            else  return true;
}
bool operator>=(BigNumber AfterNumber){
            BigNumber Temp;
            Temp.Length = Length;
            Temp.Number = Number;
            Temp.SigneNumber = SigneNumber;
            if(Temp == AfterNumber || Temp > AfterNumber)   return true;
            else  return false;
}
bool operator>(BigNumber AfterNumber){
            int j;
            if(SigneNumber > AfterNumber.SigneNumber) return true;
            else if(SigneNumber == AfterNumber.SigneNumber){
                if(Length > AfterNumber.Length){
                    if(SigneNumber > 0)     return true;
                    else  return false;
                }
                else if(Length < AfterNumber.Length){
                    if(SigneNumber > 0)     return false;
                    else  return true;
                }
                else{
                    for(j = 0 ; j < Length ; j++){
                        if(Number[j] > AfterNumber.Number[j]){
                            if(SigneNumber > 0)     return true;
                            else  return false;
                        }
                        else if(Number[j] < AfterNumber.Number[j]){
                            if(SigneNumber < 0)     return true;
                            else  return false;
                        }
                    }
                    return false;
                }
```

```
	}
	else  return false;
}
bool operator<=(BigNumber AfterNumber){
	BigNumber Temp;
	Temp.Length = Length;
	Temp.Number = Number;
	Temp.SigneNumber = SigneNumber;
	if(Temp == AfterNumber || Temp < AfterNumber)   return true;
	else  return false;
}
bool operator<(BigNumber AfterNumber){
	int j;
	if(SigneNumber < AfterNumber.SigneNumber) return true;
	else if(SigneNumber == AfterNumber.SigneNumber){
		if(Length < AfterNumber.Length){
			if(SigneNumber > 0)	return true;
			else  return false;
		}
		else if(Length > AfterNumber.Length){
			if(SigneNumber > 0)	return false;
			else  return true;
		}
		else{
			for(j = 0 ; j < Length ; j++){
				if(Number[j] < AfterNumber.Number[j]){
					if(SigneNumber > 0)	return true;
					else  return false;
					break;
				}
			}
			return false;
		}
	}
	else  return false;
}
void Print(){
	int i;
	if(SigneNumber < 0)	putchar('-');
	for(i = 0 ; i < Length ; i++)
		putchar(Number[i] + '0');
	if(Length <= 0)   cout << '0';
}
void Read(){
	string Input;
	cin >> Input;
	int i;
	if(Input[0] == '-'){
		Length = (Input.length() - 1);
		Number = new int[Length];
		SigneNumber = -1;
		for(i = 1 ; i < (Length + 1) ; i++)
			Number[i - 1] = (Input[i] - '0');
	}
	else{
		Length = Input.length();
		Number = new int[Length];
		SigneNumber = 1;
		for(i = 0 ; i < Length ; i++)
			Number[i] = (Input[i] - '0');
	}
	}
};
int main(){
	BigNumber a, b, c;
	char f;
	while(true){
		a.Read();
		cin >> f;
		b.Read();
		if(a == BigNumber("0") && b == BigNumber("0"))  break;
		(a+b).print();
		cout<<endl;
	}
	system("pause");
    return 0;
}
---
///////******* Difference_constraints *******///////
#include <iostream>
#include <map>
#include <list>
#include <string>
#include <queue>
using namespace std;
typedef struct edge{
	int e;
	int d;
};
list<edge> e[1000];
string name[1000];
map<string,int> h;
int dis[1000];
int time[1000];
bool hsh[1000];
int n;
int sa[1000],sb[1000],sc[1000],sp;
void init(){
```

```
	int i,j;
	n=1;
	sp=0;
	for(i=0;i<1000;i++)
		dis[i]=1000000000;
	memset(time,0,sizeof(time));
	memset(hsh,false,sizeof(hsh));
}
void spfa(){
	queue<int> q;
	list<edge>::iterator i;
	int t;
	hsh[0]=true;
	dis[0]=0;
	q.push(0);
	while(!q.empty()){
		t=q.front();	q.pop();
		hsh[t]=false;
		time[t]++;
		if(time[t]>2*n)	return;
		for(i=e[t].begin();i!=e[t].end();i++){
			if(dis[i->e]>dis[t]+i->d){
				dis[i->e]=dis[t]+i->d;
				if(!hsh[i->e]){
					hsh[i->e]=true;
					q.push(i->e);
				}
			}
		}
	}
}
bool chk(){
	int i;
	for(i=0;i<sp;i++)
		if(dis[sa[i]]-dis[sb[i]]>sc[i])	return false;
	return true;
}
int main(){
	int i,j;
	string ta,tb;
	int a,b;
	edge tmp;
	int tc;
	init();
	while(true){
		cin>>ta>>tb>>tc;
		if(cin.eof())	break;
		a=h[ta];
		if(a==0){
			a=n;	n++;
			name[a]=ta;
			h[ta]=a;
		}
		b=h[tb];
		if(b==0){
			b=n;	n++;
			name[b]=tb;
			h[tb]=b;
		}
		tmp.e=a;	tmp.d=tc;
		e[b].push_back(tmp);
		sa[sp]=a;
		sb[sp]=b;
		sc[sp++]=tc;
	}
	for(i=1;i<n;i++){
		tmp.e=i;	tmp.d=0;
		e[0].push_back(tmp);
	}
	spfa();
	if(chk()){
		for(i=1;i<n;i++)
			cout<<name[i]<<" = "<<dis[i]<<endl;
	}
	else	cout<<"imposible"<<endl;
	system("pause");
	return 0;
}
---
```