```cpp
/// Bismillahi-r-Rahmani-r-Rahim
#include<bits/stdc++.h>
using namespace std;
#define debug(args...){ string _s = #args;replace(_s.begin(),_s.end(),',', ' ');stringstream
_ss(_s);istream_iterator<string>_it(_ss);err(_it, args);} cout<<endl;
void err(istream_iterator<string> it) {}
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args) {cerr << *it << "=" << a << ", "; err(++it,
args...);}
#define ll long long int
#define MAX 2134567891
#define PF(a) cout<<a<<endl;
#define pf(a) printf("%lld", a);
#define sf(a) scanf("%lld", &a);
#define fr(i,n) for(i=0;i<n;i++)
#define rep(i,n) for(i=1;i<=n;i++)
#define rev(i,a,n) for(i=n;i>=a;i--)
#define FOR(i,a,n) for(i=a;i<=n;i++)
#define ALL(n) n.begin(),n.end()
#define mem(x,n) memset(x,n,sizeof(x));
//int fx[]={+1,-1,+0,+0};
//int fy[]={+0,+0,+1,-1};
//int fx[]={+0,+0,+1,-1,-1,+1,-1,+1}; // Kings Move
//int fy[]={-1,+1,+0,+0,+1,+1,-1,-1}; // Kings Move
//int fx[]={-2, -2, -1, -1, 1, 1, 2, 2}; // Knights Move
//int fy[]={-1, 1, -2, 2, -2, 2, -1, 1}; // Knights Move
#define TC(t) printf("Case %lld: ",t);
#define ans(t,c) printf("Case %lld: %lld\n",t,c);
#define SETP(n) cout<<setprecision(n)<<fixed;
#define READ freopen("F:\\Project\\Test_Case.txt","r",stdin)
#define WRITE freopen("F:\\Project\\Output_Test.txt","w",stdout)
#define IO ios_base::sync_with_stdio(0); cin.tie(0);cout.tie(0);
#define PAIR pair<ll,ll>
#define MP make_pair
#define pb push_back
#define eb emplace_back
#define ff first
#define ss second
#define NL printf("\n");
#define bug(a) cout<<#a<<" "<<a<<" ";
#define hlw printf("hlw\n");
#define hii printf("hii\n");
#define NN 111
#define MOD (ll)1e9+7 /// 10^9+7
#define N (ll)1e6+7 ///10^6->6 zero after 1 **
ll x[N],y[N],z[N],n;
string s,S;
vector<ll>v;
//bitset<N>B;
//map <ll,ll> mp;
/// priority_queue<ll, vector<ll>, greater<ll> > pq;
int main()
{
 //IO;
 //while(1)
 //READ;WRITE;
{
 ll a=0,b=0,c=0,d,e,f,g,i,j,k,l,m,p,q,r,u,w,t,tc=1;
 ll in,loc,val,sz,lo,hi,mid,mn=MAX,mx=0,sum=0,ans=0;
//cin>>tc;
rep(t,tc)
{

}
}
 return 0;
}
/// Division MOD needs BigMod(a,n-2)
////////////////////////////////////////////////// STRING/////////////////////////////////////////
unsigned bernstein_hash ( void *key, int len )
{
 unsigned char *p = key;
 unsigned h = 0;
 int i;
 for ( i = 0; i < len; i++ )
 h = 33 * h + p[i];
 return h;
}
/// string matching
vector<int> rabin_karp_HASH(string const& s, string const& t) {
 const int p = 31;
 const int m = 1e9 + 9;
```

```
 int S = s.size(), T = t.size();
 vector<long long> p_pow(max(S, T));
 vector<long long> h(T + 1, 0);
 long long h_s = 0;
 vector<int> occurences;
 p_pow[0] = 1;
 for (int i = 1; i < (int)p_pow.size(); i++)
 p_pow[i] = (p_pow[i-1] * p) % m;
 for (int i = 0; i < T; i++)
 h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
 for (int i = 0; i < S; i++)
 h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;
 for (int i = 0; i + S - 1 < T; i++)
 {
 long long cur_h = (h[i+S] + m - h[i]) % m;
 if (cur_h == h_s * p_pow[i] % m)
 occurences.push_back(i);
 }
 return occurences;
}
/// KMP with LPS (find pattern)
void LPS()
{
 ll i,j,l=pat.size();
 i=0,j=-1;
 lps[i]=j;
 while(i<l)
 {
 while(pat[i]!=pat[j] && j>=0)
 j=lps[j];
 i++,j++;
 lps[i]=j;
 }
}
ll KMP(string txt)
{
 pat=txt; reverse(ALL(pat));
 LPS(pat);
 ll i,j,n,m;
 n=txt.size();
 m=pat.size();
 i=j=0;
 while(i<n)
 {
 while(j>=0 && txt[i]!=pat[j])
 j=lps[j];
 i++,j++;
 }
 return j;
}
string sub_pal(string s) /// Find Prefix Sub_Palindrome Linear
{
 string a = s;
 reverse(a.begin(), a.end());
 a = s + "#" + a;
 //cout<<a<<endl;
 ll c = 0,pref[99]={0};
 for (int i = 1; i < (int)a.size(); i++)
 {
 cout<<"C "<<c<<endl;
 while (c != 0 && a[c] != a[i])
 c = pref[c - 1];
 if (a[c] == a[i])
 c++;
 pref[i] = c;
 }
 return s.substr(0, c);
}
string Manacher(string s) /// longest subpalindrome
{
 string T="#";// Transform S to T
 for(int i=0;i<s.size();i++)
 T+=s.substr(i,1)+"#";
 int P[T.size()+5]={0}; // Array to record longest palindrome
 int center=0,boundary=0,maxLen=0,resCenter=0;
 for(int i=1;i<T.size()-1;i++)
 {
 int iMirror=2*center-i; // calc mirror i = center-(i-center)
 if(i<boundary)
 P[i]=min(boundary-i,P[iMirror]);
 while(i-1-P[i]>=0 && i+1+P[i]<=T.size()-1 && T[i+1+P[i]]==T[i-1-P[i]]) // Attempt to
expand palindrome centered at i
```

```
 P[i]++;
 if(i+P[i]>boundary)
 { // update center and boundary
 center = i;
 boundary = i+P[i];
 }
 if(P[i]>maxLen)
 { // update result
 maxLen = P[i];
 resCenter = i;
 }
 }
 return s.substr((resCenter - maxLen)/2, maxLen);
}
vector<int>z_algo(string s) /// finds all occurrences of a pattern linear
{
 int i,l,r,n;
 n=s.length();
 vector<int> z(n);
 for (i = 1, l = 0, r = 0; i < n; ++i)
 {
 if (i <= r)
 z[i] = min (r - i + 1, z[i - l]);
 while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ///Checking character and ++1
 ++z[i];
 if (i + z[i] - 1 > r)
 l = i, r = i + z[i] - 1;
 }
 return z;
}
//////////////////////Sparse Table
ll st[22][N],x[N],logs[N];
void build(ll n)/// 0
{
 ll i,j,k;
 logs[1]=0; for(i=2;i<=n;i++)logs[i]= logs[i/2]+1;
 for(i=0;i<n;i++)st[0][i]=x[i];
 for(i=1; (1<<i) <n; i++)
 {
 for(j=0; j+(1<<i)<=n; j++) /// 1<<i = current_len
 {
 st[i][j]=min(st[i-1][j], st[i-1][j + (1<<i-1)]);
 }
 }
}
ll query(ll l, ll r)
{
 ll pow = logs[r-l+1]; //log2(r-l+1);
 return min(st[pow][l], st[pow][r-(1<<pow)+1]);
}
build(n); cout<<query(l,r)<<endl;
////////////////////////////////Segment Tree/////////////////////////////////////////////
ll tree[4*N],tr[N],lazy[4*N];
void build(ll in,ll L,ll R)
{
 if(L==R)
 {
 tree[in]=tr[L];
 return;
 }
 ll mid=(L+R)/2;
 build(in*2,L,mid);
 build(in*2+1,mid+1,R);
 tree[in]=min(tree[in*2],tree[in*2+1]); /// Change Function
}
void lazy_update (ll in,ll L,ll R,ll x,ll y,ll val)
{
 if(x>y)return;
 if(lazy[in]!=0)
 {
 tree[in]+=lazy[in];
 if(L!=R)
 {
 lazy[in*2]+=lazy[in];
 lazy[in*2+1]+=lazy[in];
 }
 lazy[in]=0;
 }
 if(x>R || y<L)return;
 if(x<=L && y>=R)
 {
 tree[in]+=val;
```

```
 if(L!=R)
 {
 lazy[in*2]+=val;
 lazy[in*2+1]+=val;
 }
 return;
 }
 ll mid=(L+R)/2;
 lazy_update(in*2,L,mid,x,y,val);
 lazy_update(in*2+1,mid+1,R,x,y,val);
 tree[in]=tree[in*2]+tree[in*2+1];
}
ll lazy_query(ll in,ll L,ll R,ll x,ll y)
{
 if(x>y)return 0;
 if(lazy[in]!=0)
 {
 tree[in]+=lazy[in];
 if(L!=R)
 {
 lazy[in*2]+=lazy[in];
 lazy[in*2+1]+=lazy[in];
 }
 lazy[in]=0;
 }
 if(x>R || y<L)return 0;
 if(x<=L && y>=R)
 return tree[in];
 ll p,q,mid=(L+R)/2;
 p=lazy_query(in*2,L,mid,x,y);
 q=lazy_query(in*2+1,mid+1,R,x,y);
 return p+q;
}
void update(ll in,ll L,ll R,ll pos,ll val)
{
 if(pos>R||L>pos)return;
 if(L==R&&pos==L)
 {
 tree[in]+=val; /// Change Function
 return;
 }
 ll mid=(L+R)/2;
 update(in*2,L,mid,pos,val);
 update(in*2+1,mid+1,R,pos,val);
 tree[in]=tree[in*2]+tree[in*2+1]; /// Change Function
}
ll query(ll L,ll R,ll in,ll i,ll j)
{
 if(j<L||i>R)return MAX;
 //return 0;
 if(L>=i&&j>=R)return tree[in];
 ll p,q,mid=(L+R)/2;
 p=query(L,mid,in*2,i,j);
 q=query(mid+1,R,in*2+1,i,j);
 return min(p,q); /// Change Function
}
build(1,1,n); cout<<query(1,n,1,a,b)<<endl;
lazy_update(1,1,n,a,b,c); cout<<lazy_query(1,n,1,a,b)<<endl;
////////////////////////////////DP/////////////////////////////
ll LCS(char p[],char q[],int a,int b)
{
 ///All loop will work through 1 to n/m here...
 int i,j,k;
 rep(i,a)
 x[i][0]=0;
 rep(i,b)
 x[0][i]=0;
 rep(i,a)
 rep(j,b)
 {
 if(p[i]==q[j])x[i][j]=x[i-1][j-1]+1;
 else x[i][j]=max(x[i][j-1],x[i-1][j]);
 }
 return x[a][b];
}
ll LIS(ll n)
{
 ll i,a,in=0,st,en,mid,ans=-1;
 ar[1]=INT_MIN;
 rep(i,n)
 {
 a=x[i];
```

```
 if(in==0 || a>ar[in])
 {
 cout<<"Appending "<<a<<" in "<<1+in<<endl;
 ar[++in]=a;
 }
 else if(a<x[1])
 ar[1]=a;
 else
 {
 st=1,en=in;
 while(st<=en)
 {
 mid=(st+en)/2;
 if(ar[mid]<a)
 st=mid+1;
 else en=mid-1;
 }
 ar[st]=a;
 cout<<mid<<" mid\n";
 }
 cout<<"i "<<i<<" a "<<a<<" in "<<in<<endl;
 }
 return in;
}
////////////////////////////BIT/////////////////////////////////////
void update(ll pos,ll val)
{
 while(pos<=n)
 {
 x[pos]+=val;
 pos+=(pos & -pos);
 }
}
ll query(ll pos)
{
 ll sum=0;
 while(pos)
 {
 sum+=x[pos];
 pos-=(pos & -pos);
 }
 return sum;
}
 rep(i,n)
 {cin>>a; update(i,a); /// 1-based}
 cout<<query(4)<<" "<<query(2)<<" Ans "<<query(4)-query(2)<<endl;
////////////////////////////////////MATH/////////////////////////////////////
ll spf[N]; vector<ll>primes;
void sieve() ///with SPF
{
 for(int i = 2; i < N; i++)
 {
 if (spf[i] == 0) spf[i] = i, primes.push_back(i);
 int sz = primes.size();
 for (int j=0; j<sz && i*primes[j]<N && primes[j]<=spf[i]; j++)
 {
 spf[i * primes[j]] = primes[j];
 }
 }
}
ll nCr(ll n,ll r) /// nCr DP
{
 ll &ret=dp[n][r];
 if(~ret)return ret;
 if(n==r)return ret=1;
 if(r==1)return ret=n;
 return ret=nCr(n-1,r)+nCr(n-1,r-1);
}
ll bigmod(ll n,ll p,ll MOD) /// finds n ^ p % MOD
{
 if(p==0)return 1;
 ll x=bigmod(n,p/2,MOD);
 x=(x*x)%MOD;
 if(p%2)x=(x*n)%MOD;
 return x;
}
ll precal_nCr(ll n, ll r) /// larger inputs and MOD required
{
 /// Precal Starts Here
 fact[1] = 1;
 for(ll i=2; i<n; i++) fact[i] = (i*fact[i-1])%MOD;
 invfact[n-1] = bigmod(fact[n-1], MOD-2, MOD);
```

```
 for (ll i=n-2; i>=0; i--) invfact[i] = (invfact[i+1]*(i+1))%MOD;
 /// Precal Ends Here
 if (r<0 || r>n) return 0;
 return (fact[n]*(invfact[r]*invfact[n-r])%MOD)%MOD;
}
void permutation(string s,int i,int n)
{
 if(i==n){cout<<s<<endl;return ;}
 for(int j=i;j<=n;j++)
 {
 swap(s[i],s[j]);
 permutation(s,i+1,n);
 }
}
ll mod_inverse(ll a,ll mod)
{
 return bigmod(a,mod-2,mod);
}
void allPossibleSubset(int n)
{
for(ll mask = 0; mask < (1 << n); mask++) {
 ll sum_of_this_subset = 0;
 for(int i = 0; i < n; i++)
 {
 if(mask & (1 << i)) {
 sum_of_this_subset += x[i];
 }
 }
 }
}
/// Find numbers of co-prime of N which are less than N
void totient()
{
 ll i,j,k;
 for(i=1;i<=N;i++)phi[i]=i;
 for(i=2;i<=N;i++)
 {
 if(phi[i]==i)
 {
 for(j=i;j<=N;j+=i)
 {
 phi[j]= (phi[j]*(i-1))/i;
 }
 }
 }
}
/// Find eulerphi for any numbers with prime pre-calculated
int eulerPhi ( int n ) {
 int res = n;
 int sqrtn = sqrt ( n );
 for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
 if ( n % prime[i] == 0 ) {
 while ( n % prime[i] == 0 ) {
 n /= prime[i];
 }
 sqrtn = sqrt ( n );
 res /= prime[i];
 res *= prime[i] - 1;
 }
 }
 if ( n != 1 ) {
 res /= n;
 res *= n - 1;
 }
 return res;
}
ll binarySearch(ll lo,ll hi,ll key)
{
 while(lo<=hi)
 {
 ll mid=(lo+hi)/2;
 if(x[mid]==key)
 {
 ll ans=mid;
 lo=mid+1;
 }
 else
 hi=mid-1;
 }
}
int gcd(int a,int b)
{
```

```
 while(b)
 a %= b, swap(a, b);
 return a;
}
ll Inclusion_Exclusion()
{
 ll a=0,b,c=0,cnt,i,j,k,m,n;
 cnt=pow(2,m);
 rep(i,cnt-1)
 {
 a=1;
 fr(j,m)
 {
 if(i & 1<<j)
 a=(a*x[j])/__gcd(a,x[j]);
 }
 a=n/a;
 b=__builtin_popcountll(i);
 if(b%2)c+=a;
 else c-=a;
 }
 return n-c;
}
double Angle(double Ax,double Ay,double Bx,double By,double Cx,double Cy)
{
 double a1,a2,b1,b2,u,v,p,ang;
 a1=Ax-Bx; b1=Ay-By;
 a2=Cx-Bx; b2=Cy-By;
 p=a1*a2+b1*b2;
 u=sqrt(a1*a1+b1*b1);
 v=sqrt(a2*a2+b2*b2);
 ang = acos(p/(u*v));
 return (ang*180)/acos(0.0);
}
 ///Calculate Time Complexity
 clock_t t1,t2; double t;
 t1=clock();
 fr(i,10000)fr(j,10000)x[i]=rand();
 t2=clock();
 t=(t2-t1)/(CLOCKS_PER_SEC);
 cout<<"Time: "<<t<<endl;
////////////////////////////////////// LCA by Sparse Table//////////////////////////
void walk(ll s, ll d)
{
 ll i, in;
 last[s]=k;
 nodes[k]=s;
 depth[k++]=d;
 fr(i,v[s].size())
 {
 in=v[s][i];
 if(vis[in])continue;
 vis[in]=1;
 walk(in,d+1);
 nodes[k]=s;
 depth[k++]=d;
 }
}
void sparse_table(ll n)/// 0 based indexing
{
 ll node_a,node_b,i,j,k;
 for(i=0;i<n;i++)st[0][i]=i; /// storing nodes, not values
 for(i=1; (1<<i) <n; i++)
 {
 for(j=0; j+(1<<i)<=n; j++) /// 1<<i = current_len
 {
 node_a=st[i-1][j];
 node_b=st[i-1][j + (1<<i-1)];
 st[i][j] = depth[node_a]<=depth[node_b]? node_a:node_b; /// For RMQ
 }
 }
}
ll LCA(ll l,ll r)
{l=last[l],r=last[r];if(l>r)swap(l,r);ll pow = log2(r-l+1);ll a,b;a=st[pow][l]; b=st[pow][r-(1<<pow)+1];return nodes[deptl
int main(){ ///0 based indexing
 vis[0]=1;
 walk(0,0);
 sparse_table(2*n-1);
cin>>a>>b; cout<<LCA(a-1,b-1)+1<<endl;
}
/* author : s@if */
#include<bits/stdc++.h>
```

```
#include<ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;
#define         NIL             -1
#define         INF             1e9
#define         EPS             1e-9
#define         SAIF            main
#define         fi              first
#define         sec             second
#define         MAX             INT_MAX
#define         ll              long long
#define         PI              acos(-1.0)
#define         MOD             1000000007
#define         PLL             pair<ll,ll>
#define         PII             pair<int,int>
#define         ull             unsigned long long
#define         For(i,a,b)      for(int i=a;i<=(int)b;i++)
typedef tree<int, null_type, less<int>, rb_tree_tag,
            tree_order_statistics_node_update> new_data_set;
//*find_by_order(k)  gives the kth element;
//order_of_key(item)   gives the index(number of element strictly less than item) of item;
inline int in() {int x; scanf("%d", &x); return x; }
bool Check(int N , int pos) {   return (bool) (N & (1<<pos));}
int Set(int N, int pos) {   return N = N | (1<<pos);}
int fx[]={+0,+0,+1,-1,-1,+1,-1,+1};    // King's move
int fy[]={-1,+1,+0,+0,+1,+1,-1,-1};
int hx[]={-2,-2,-1,+1,+2,+2,-1,+1};    // Knight's move
int hy[]={+1,-1,+2,+2,-1,+1,-2,-2};
int dx[]={+1,-1,+0,+0};
int dy[]={+0,+0,+1,-1};
const int MAXN = (int)2e5+9;
/*
// Hashing
ll base = 247, M = 1000000007;
ll Hash[MAXN], power[MAXN], L;
void init(void)
{
    power[0] = 1; Hash[0] = 0;
    for(int i=1; i<MAXN; i++)
    {
        power[i] = (power[i-1]*base)%M;
    }
}
void Hashing(string s)
{
    L = s.size();
    ll h = 0;
    for(int i=1; i<=L; i++)
    {
        ll tmp = (h*base)%M;
        tmp = (tmp+s[i-1]-'a'+1)%M;
        Hash[i] = h = tmp;
    }
    return;
}
ll HashOf(string p)
{
    int l = p.size();
    ll h = 0;
    for(int i=1; i<=l; i++)
    {
        ll tmp = (h*base)%M;
        tmp = (tmp+p[i-1]-'a'+1)%M;
        h = tmp;
    }
    return h;
}
ll HashOfSubstring(int l, int r)
{
    ll a, b, ret;
    a = Hash[l-1], b = Hash[r];
    a = (a*power[r-l+1])%M;
    ret = (b-a+M)%M;

    return ret;
}
int FindPattern(string p)
{
    int i, l = p.size();
    ll h1 = HashOf(p);

    for(i=1; i<=L-l+1; i++)
```

```
    {
        int x = i, y = i+l-1;
        ll h2 = HashOfSubstring(x, y);
        if(h1==h2) return i-1;
    }

    return -1;
}
*/
/*
// trie
struct node
{
    bool mark;
    node *next[30];
    node()
    {
        mark=false;

        for(int i=0;i<26;i++)
        {
            next[i]=NULL;
        }
    }
};
node *root;
void add(string s)
{
    int l=s.size();

    node *curr=root;
    for(int i=0;i<l;i++)
    {
        int id=s[i]-'a';
        if(curr->next[id]==NULL)
            curr->next[id]=new node();

        curr=curr->next[id];
    }

    curr->mark=true;
}
bool _search(string s)
{
    int l=s.size();

    node *curr=root;
    for(int i=0;i<l;i++)
    {
        int id=s[i]-'a';
        if(curr->next[id]==NULL)
            curr->next[id]=new node();
        curr=curr->next[id];
    }
    return curr->mark;
}
void del(node *curr)
{
    for(int i=0;i<26;i++)
    {
        if(curr->next[i])
            del(curr->next[i]);
    }
    delete(curr);
}
*/
/*
// KMP
void kmp(string T, string P)
{
    int n=strlen(T);
    int m=strlen(P);
    int pi[m+9], i, now;

    now=pi[0]=-1;
    for(i=1;i<m;i++)
    {
        while(now!=-1 && P[now+1]!=P[i])
        {
            now=pi[now];
        }
        if(P[now+1]==P[i])
```

```
            pi[i]= ++now;
        else
            pi[i]=now=-1;
    }
    int cnt=0;
    now=-1;
    for(i=0;i<n;i++)
    {
        while(now!=-1 && P[now+1]!=T[i])
        {
            now=pi[now];
        }
        if(P[now+1]==T[i])
            now++;
        else
            now=-1;

        if(now==m-1)
        {
            cnt++;
            now=pi[now];

        }
    }
    printf("Case %d: %d\n",++t,cnt);

    return;
}*/
/*
// Articulation Point
int vis[MAXN], d[MAXN], low[MAXN], art[MAXN], Tm;
vector<int>adj[MAXN];
void init(int n)
{
    for(int i=0; i<=n; i++)
    {
        vis[i] = 0; art[i] = 0, Tm = 0;
        adj[i].clear();
    }
}
void find_articulation_point(int u)
{
    Tm++; d[u] = low[u] = Tm;
    vis[u] = 1; int child = 0;

    for(int i=0; i<adj[u].size(); i++)
    {
        int v = adj[u][i];

        if(vis[v]==1)
        {
            low[u] = min(low[u], d[v]);
        }
        else
        {
            child++;
            find_articulation_point(v);
            low[u] = min(low[u], low[v]);
            if(d[u]<=low[v] && u!=1) art[u] = 1;
        }
    }
    if(u==1 && child>1) art[u] = 1;
}
*/
/*
// SCC
vector<int>component[MAXN];
vector<int>g[MAXN];
vector<int>rev[MAXN];
stack<int>stk;
int n,mark;
int vis[MAXN];
void dfs1(int cur)
{
    vis[cur]=1;

    for(int i=0;i<g[cur].size();i++)
    {
        int v=g[cur][i];

        if(!vis[v])
            dfs1(v);
```

```
    }

    stk.push(cur);
}
void dfs2(int cur,int mark)
{
    vis[cur]=1;
    component[mark].push_back(cur);
    for(int i=0;i<rev[cur].size();i++)
    {
        int v=rev[cur][i];

        if(!vis[v])
        {
            dfs2(v,mark);
        }
    }
}
void SCC(void)
{
    cin>>n>>m;
    while(m--)
    {
        cin>>u>>v;

        g[u].push_back(v);
        rev[v].push_back(u);
    }
    memset(vis,0,sizeof(vis));
    for(i=1;i<=n;i++)
        if(!vis[i])
            dfs1(i);

    memset(vis,0,sizeof(vis));
    mark=0;
    while(!stk.empty())
    {
        u=stk.top();
        stk.pop();

        if(!vis[u])
        {
            dfs2(u,++mark);
        }
    }
    for(i=1;i<=mark;i++)
    {
        cout<<"component "<<i<<" : ";

        for(j=0;j<component[i].size();j++)
        cout<<component[i][j]<<" ";
        cout<<endl;
    }
    cout<<endl;
}
*/
/*
//LCA
int L[mx];
int P[mx][22];
int T[mx];
vector<int>g[mx];
void dfs(int from,int u,int dep)
{
    T[u]=from;
    L[u]=dep;
    for(int i=0;i<(int)g[u].size();i++)
    {
        int v=g[u][i];
        if(v==from) continue;
        dfs(u,v,dep+1);
    }
}
int lca_query(int N, int p, int q)
{
    int tmp, log, i;

    if (L[p] < L[q])
        tmp = p, p = q, q = tmp;
    log=1;
    while(1) {
        int next=log+1;
```

```
        if((1<<next)>L[p])break;
        log++;

    }
    for (i = log; i >= 0; i--)
        if (L[p] - (1 << i) >= L[q])
            p = P[p][i];

    if (p == q)
        return p;

    for (i = log; i >= 0; i--)
        if (P[p][i] != -1 && P[p][i] != P[q][i])
            p = P[p][i], q = P[q][i];

    return T[p];
}
void lca_init(int N)
{
    memset (P,-1,sizeof(P));
    int i, j;
    for (i = 0; i < N; i++)
        P[i][0] = T[i];

    for (j = 1; 1 << j < N; j++)
        for (i = 0; i < N; i++)
            if (P[i][j - 1] != -1)
                P[i][j] = P[P[i][j - 1]][j - 1];
}
*/
/*
// Discrete Logarithm

ll Discrete_Log(ll a, ll b, ll m)
{
    if(a==0)
    {
        if(b==0) return 1;
        else return -1;
    }
    a%=m, b%=m; ll g, k = 1, add = 0;
    while((g=__gcd(a,m))>1)
    {
        if(b==k) return add;
        if(b%g) return -1;

        b/=g, m/=g; ++add;

        k = (k*a/g)%m;
    }
    map<ll,ll>Map; ll n = sqrt(m)+1;

    for(ll q=0, curr=b; q<=n; q++)
    {
        Map[curr] = q;
        curr = (curr*a)%m;
    }
    ll an = 1;
    for(ll p=1; p<=n; p++)
        an = (an*a)%m;
    for(ll p=1, curr=k; p<=n; p++)
    {
        curr = (curr*an)%m;

        if(Map[curr])
            return n*p-Map[curr]+add;
    }
    return -1;
}
*/
void solve(void)
{
    ll a, b, i,j,k,l,m,n,p,q,x,y,u,v,w,r,tc,t;
    return;
}
int SAIF()
{
    int tc, t = 0;
    cin>>tc; while(tc--) solve();
    return 0;
}
```

```
// read the question correctly (is y a vowel? what are the exact constraints?)
// look out for SPECIAL CASES (n=1?) and overflow (ll vs int?)
```