Table of Contents

Precode

```
#include <cstdio>
#include <sstream>
#include <cstdlib>
#include <cctype>
#include <cmath>
#include <algorithm>
#include <set>
#include <queue>
#include <stack>
#include <list>
#include <iostream>
#include <fstream>
#include <numeric>
#include <string>
#include <vector>
#include <cstring>
#include <map>
#include <iterator>
#include<complex>
//#include <bits/stdc++.h>

#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("Ofast")
#pragma GCC
target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
#pragma GCC optimize("unroll-loops")

using namespace std;

#define HI            printf("HI\n")
#define sf            scanf
#define pf            printf
#define sf1(a)        scanf("%d",&a)
#define sf2(a,b)      scanf("%d %d",&a,&b)
#define sf3(a,b,c)    scanf("%d %d %d",&a,&b,&c)
#define sf4(a,b,c,d)  scanf("%d %d %d %d",&a,&b,&c,&d)
#define sf1ll(a)      scanf("%lld",&a)
#define sf2ll(a,b)    scanf("%lld %lld",&a,&b)
#define sf3ll(a,b,c)  scanf("%lld %lld %lld",&a,&b,&c)
#define sf4ll(a,b,c,d)  scanf("%lld %lld %lld %lld",&a,&b,&c,&d)
#define pb            push_back
#define ppb           pop_back
#define ppf           push_front
#define popf          pop_front
#define ll            long long int
#define ui            unsigned int
#define ull           unsigned long long
#define fs            first
#define sc            second
#define clr( a, b )   memset((a),b,sizeof(a))
#define jora          pair<int, int>
#define jora_d        pair<double, double>
#define jora_ll       pair<long long int, long long int>
#define mp            make_pair
```

```
#define max3(a,b,c)     max(a,max(b,c))
#define min3(a,b,c)     min(a,min(b,c))
#define PI              acos(-1.0)
#define ps              pf("PASS\n")
#define popc(a)         (__builtin_popcount(a))

template<class T1> void deb(T1 e1) {
   cout<<e1<<endl;
}
template<class T1,class T2> void deb(T1 e1,T2 e2) {
   cout<<e1<<" "<<e2<<endl;
}
template<class T1,class T2,class T3> void deb(T1 e1,T2 e2,T3 e3) {
   cout<<e1<<" "<<e2<<" "<<e3<<endl;
}
template<class T1,class T2,class T3,class T4> void deb(T1 e1,T2 e2,T3
e3,T4 e4) {
   cout<<e1<<" "<<e2<<" "<<e3<<" "<<e4<<endl;
}
template<class T1,class T2,class T3,class T4,class T5> void deb(T1 e1,T2
e2,T3 e3,T4 e4,T5 e5) {
   cout<<e1<<" "<<e2<<" "<<e3<<" "<<e4<<" "<<e5<<endl;
}
template<class T1,class T2,class T3,class T4,class T5,class T6> void deb(T1
e1,T2 e2,T3 e3,T4 e4,T5 e5,T6 e6) {
   cout<<e1<<" "<<e2<<" "<<e3<<" "<<e4<<" "<<e5<<" "<<e6<<endl;
}

/// <---------------------------  For Bitmasking  ------------------------------->
//int on( int n, int pos ){
//    return n = n|( 1<<pos );
//}
//bool check( int n, int pos ){
//    return (bool)( n&( 1<<pos ) );
//}
//int off( int n, int pos ){
//    return n = n&~( 1<<pos );
//}
//int toggle( int n, int pos ){
//    return n = n^(1<<pos);
//}
//int count_bit( int n ){
//    return __builtin_popcount( n );
//}
/// <---------------------------  End of Bitmasking  ------------------------------->


/// <---------------------------- For B - Base Number System ----------------------------
--------->
//int base;
//int pw[10];
//void calPow(int b){
//    base = b;
//    pw[0] = 1;
//    for( int i = 1; i<10; i++ ){
//        pw[i] = pw[i-1]*base;
//    }
//}
//int getV(int mask, int pos){
//    mask /= pw[pos];
//    return ( mask%base );
//}
```

```
//int setV(int mask, int v, int pos){
//    int rem = mask%pw[pos];
//    mask /= pw[pos+1];
//    mask = ( mask*base ) + v;
//    mask = ( mask*pw[pos] ) + rem;
//    return mask;
//}
/// <---------------------------- End B - Base Number System ------------------------
---------->


// moves

//int dx[]= {0,0,1,-1};/*4 side move*/
//int dy[]= {-1,1,0,0};/*4 side move*/
//int dx[]= {1,1,0,-1,-1,-1,0,1};/*8 side move*/
//int dy[]= {0,1,1,1,0,-1,-1,-1};/*8 side move*/
//int dx[]={1,1,2,2,-1,-1,-2,-2};/*night move*/
//int dy[]={2,-2,1,-1,2,-2,1,-1};/*night move*/

//double Expo(double n, int p) {
//        if (p == 0)return 1;
//        double x = Expo(n, p >> 1);
//        x = (x * x);
//        return ((p & 1) ? (x * n) : x);
//}

//ll bigmod(ll a,ll b,ll m){if(b == 0) return 1%m;ll x = bigmod(a,b/2,m);x =
(x * x) % m;if(b % 2 == 1) x = (x * a) % m;return x;}
//ll BigMod(ll B,ll P,ll M){ ll R=1%M; while(P>0)
{if(P%2==1){R=(R*B)%M;}P/=2;B=(B*B)%M;} return R;} /// (B^P)%M
```

```
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;

#define MXN 50
#define MXE
#define MXQ
#define SZE
#define MOD
#define EPS
#define INF 1000000000
#define MX  200
#define inf 100000000

const int mod = 1000000007;

/// Policy Based DS
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace __gnu_pbds;
/// Policy Based DS

typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> set_t;

pll extended_euclid(ll a, ll b){ // returns x, y | ax + by = gcd(a,b)
   if(b == 0)    return pll(1, 0);
   else{
      pll d = extended_euclid(b, a%b);
      return pll(d.second, d.first - d.second*(a/b));
```

```
    }
}
ll modular_inverse(ll a, ll m) {
    pll ret = extended_euclid(a, m);
    return ((ret.first%m)+m)%m;
}

//Fast Reader
template<class T>inline bool read(T &x){
    int c=getchar();int sgn=1;
    while(~c&&c<'0'||c>'9'){if(c=='-')sgn=-1;c=getchar();}
    for(x=0;~c&&'0'<=c&&c<='9';c=getchar())x=x*10+c-'0';
    x*=sgn; return ~c;
}

int main() {
//mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
    //ios_base::sync_with_stdio(0);
//    freopen("input.txt", "r", stdin);


    return 0;
}
```

Graph Theory

1. Dinic's-Maxflow

```
///V^2*E Complexity
///number of augment path * (V+E)
///Base doesn't matter

const int INF = 2000000000;
const int MAXN = 100;///total nodes
const int MAXM = 10000;///total edges

int N,edges;
int last[MAXN],Prev[MAXM],head[MAXM];
int Cap[MAXM],Flow[MAXM];
int dist[MAXN];
int nextEdge[MAXN];
        ///used for keeping track of next edge of ith node

queue<int> Q;

void init(int N) {
   edges=0;
   memset(last,-1,sizeof(int)*N);
}
//cap=capacity of edges , flow = initial flow
inline void addEdge(int u,int v,int cap,int flow) {
   head[edges]=v;
   Prev[edges]=last[u];
   Cap[edges]=cap;
   Flow[edges]=flow;
   last[u]=edges++;

   head[edges]=u;
   Prev[edges]=last[v];
   Cap[edges]=0;
   Flow[edges]=0;
   last[v]=edges++;
}
inline bool dinicBfs(int S,int E,int N) {
   int from=S,to,cap,flow;
   memset(dist,0,sizeof(int)*N);
   dist[from]=1;
   while(!Q.empty()) Q.pop();
   Q.push(from);
   while(!Q.empty()) {
      from=Q.front();
      Q.pop();
      for(int e=last[from]; e>=0; e=Prev[e]) {
         to=head[e];
         cap=Cap[e];
         flow=Flow[e];
         if(!dist[to] && cap>flow) {
            dist[to]=dist[from]+1;
            Q.push(to);
         }
      }
   }
   return (dist[E]!=0);
}
inline int dfs(int from,int minEdge,int E) {
   if(!minEdge) return 0;
```

```
    if(from==E) return minEdge;
    int to,e,cap,flow,ret;
    for(; nextEdge[from]>=0; nextEdge[from]=Prev[e]) {
        e=nextEdge[from];
        to=head[e];
        cap=Cap[e];
        flow=Flow[e];
        if(dist[to]!=dist[from]+1) continue;
        ret=dfs(to,min(minEdge,cap-flow),E);
        if(ret) {
            Flow[e]+=ret;
            Flow[e^1]-=ret;
            return ret;
        }
    }
    return 0;
}
int dinicUpdate(int S,int E) {
    int flow=0;
    while(int minEdge = dfs(S,INF,E)) {
        if(minEdge==0) break;
        flow+=minEdge;
    }
    return flow;
}
int maxFlow(int S,int E,int N) {
    int totFlow=0;
    while(dinicBfs(S,E,N)) {
        /// update last edge of ith node
        for(int i=0; i<=N; i++) nextEdge[i]=last[i];
        totFlow+=dinicUpdate(S,E);
    }
    return totFlow;
}
```

2. MincostMaxFlow – SPFA

```
///V*E^2 Complexity
///number of augment path * (V+E)
///Base doesn't matter

const int MAXN = 350;          ///total nodes
const int MAXM = 120200;       ///total edges
const int oo = 120200;         ///total edges

int edges;        ///edge info
int Last[MAXN];
int Prev[MAXM],Head[MAXM];
int Cap[MAXM];
int Cost[MAXM];

int Flow[MAXN];
int edgeNo[MAXN];
int dist[MAXN];
int par[MAXN];
bool visited[MAXN];

void init(int N) {
    memset(Last,-1,sizeof(int)*N);
    edges=0;
}

void addEdge(int u,int v,int cap,int cost) {
```

```
    Head[edges]=v;
    Prev[edges]=Last[u];
    Cap[edges]=cap;
    Cost[edges]=cost;
    Last[u]=edges++;

    Head[edges]=u;
    Prev[edges]=Last[v];
    Cap[edges]=0;
    Cost[edges]=-cost;
    Last[v]=edges++;
}

queue<int> Q;
pair<int,int> SPFA(int S,int E,int N) { //source,destination,number of
nodes (give more for safety)
    int totFlow=0,totCost=0;
    while(!Q.empty()) Q.pop();
    int u,v,cap,cost;
    while(true) {
        Flow[S]=oo;
        for(int i = 0; i <= N; i++)
            dist[i] = oo;
        dist[S]=0;
        memset(visited,false,sizeof(bool)*N);
        visited[S]=1;
        Q.push(S);
        while(!Q.empty()) {
            u=Q.front();
            Q.pop();
            visited[u]=false;
            for(int e=Last[u]; e>=0; e=Prev[e]) {
                v=Head[e];
                cap=Cap[e];
                cost=Cost[e];
                if(cap&&dist[v]>dist[u]+cost) {
                    dist[v]=dist[u]+cost;
                    Flow[v]=min(Flow[u],cap);
                    edgeNo[v]=e;
                    par[v]=u;
                    if(!visited[v]) {
                        visited[v]=true;
                        Q.push(v);
                    }
                }
            }
        }
        if(dist[E]==oo) break;
        totCost+=dist[E]*Flow[E];
        totFlow+=Flow[E];
        for(int i=E; i!=S; i=par[i]) {
            Cap[edgeNo[i]]-=Flow[E];
            Cap[edgeNo[i]^1]+=Flow[E];
        }
    }
    return make_pair(totFlow,totCost);
}
```

3. All Pair Max Flow

```
///V^2*E Complexity
///number of augment path * (V+E)
///Base doesn't matter
```

```
const int INF = 200000000;
const int MAXN = 900;///total nodes
const int MAXM = 100000;///total edges

int N,edges;
int last[MAXN],preve[MAXM],head[MAXM];
int Cap[MAXM],Flow[MAXM];
int dist[MAXN];
int nextEdge[MAXN];///used for keeping track of next edge of ith node

queue<int> Q;

void init(int N)
{
   edges=0;
   memset(last,-1,sizeof(int)*N);
}

//cap=capacity of edges , flow = initial flow
inline void addEdge(int u,int v,int cap,int flow)
{
   head[edges]=v;
   preve[edges]=last[u];
   Cap[edges]=cap;
   Flow[edges]=flow;
   last[u]=edges++;

   head[edges]=u;
   preve[edges]=last[v];
   Cap[edges]=0;
   Flow[edges]=0;
   last[v]=edges++;
}

inline bool dinicBfs(int S,int E,int N)
{
   int from=S,to,cap,flow;
   memset(dist,0,sizeof dist);
   dist[from]=1;
   while(!Q.empty()) Q.pop();
   Q.push(from);
   while(!Q.empty())
   {
      from=Q.front();Q.pop();
      for(int e=last[from];e>=0;e=preve[e])
      {
         to=head[e];
         cap=Cap[e];
         flow=Flow[e];
         if(!dist[to] && cap>flow)
         {
            dist[to]=dist[from]+1;
            Q.push(to);
            ///Important
            if(to==E) return true;
            ///Need to be sure
         }
      }
   }
   return (dist[E]!=0);
}
```

```
inline int dfs(int from,int minEdge,int E)
{
   if(!minEdge) return 0;
   if(from==E) return minEdge;
   int to,e,cap,flow,ret;
   for(;nextEdge[from]>=0;nextEdge[from]=preve[e])
   {
      e=nextEdge[from];
      to=head[e];
      cap=Cap[e];
      flow=Flow[e];
      if(dist[to]!=dist[from]+1) continue;
      ret=dfs(to,min(minEdge,cap-flow),E);
      if(ret)
      {
         Flow[e]+=ret;
         Flow[e^1]-=ret;
         return ret;
      }
   }
   return 0;
}

int dinicUpdate(int S,int E)
{
   int flow=0;
   while(int minEdge = dfs(S,INF,E))
   {
      if(minEdge==0) break;
      flow+=minEdge;
   }
   return flow;
}

int maxFlow(int S,int E,int N)
{
   int totFlow=0;
   while(dinicBfs(S,E,N))
   {
      for(int i=0;i<=N;i++) nextEdge[i]=last[i];
      totFlow+=dinicUpdate(S,E);
   }
   return totFlow;
}
int vis2[205];
int ou[205][205];
vector<int>cost[205],vc[205];
void dfs2( int v,int mul,int wei)
{
   vis2[v]=1;
   ou[mul][v]=wei;
   for(int i=0;i<vc[v].size();i++)
   {
      int w=vc[v][i];
      if(vis2[w])continue;
      dfs2(w,mul,min(wei,cost[v][i]));
   }
}
int ret[205],ara[205][205],P[205];
int main()
{
```

```
int i,j,k,l,m,n,ts,casio=1;
scanf("%d",&ts);
while(ts--){
   scanf("%d",&n);
   edges=0;
   memset(last,-1,sizeof last);
   for(int i=1;i<=n;i++)
   {
      for(int j=1;j<=n;j++)
      {
         scanf("%d",&ara[i][j]);
         if(i==j)continue;
         addEdge(i,j,ara[i][j],0);
      }
   }
   for(int i=0;i<=n;i++){
         vc[i].clear();
         cost[i].clear();
         P[i]=0;
   }
   for(int i=1;i<=n;i++)
   {
    for(int j=0;j<edges;j++)Flow[j]=0;
    ret[i]=maxFlow(i,P[i],n+5);
    dinicBfs(i,-1,n);
    for(int j=i+1;j<=n;j++){
      if(dist[j]&&P[i]==P[j])P[j]=i;
    }
   }
   for(int i=1;i<=n;i++)
   {
         vc[P[i]].pb(i);
         vc[i].pb(P[i]);
         cost[i].pb(ret[i]);
         cost[P[i]].pb(ret[i]);
      }
      for(int i=1;i<=n;i++)
      {
        memset(vis2,0,sizeof vis2);
        dfs2(i,i,2147483647);
      }
      printf("Case #%d:\n",casio++);
      for(int i=1;i<=n;i++)
      {
         for(int j=1;j<=n;j++)
         {
            if(i!=j)printf("%d",ou[i][j]);
            else printf("0");
            if(j!=n)printf(" ");
         }
         printf("\n");
      }
   }
}
```

4. Dense Flow

```
///V^2*E Complexity
///number of augment path * (V+E)
///Base doesn't matter

const int INF = 20000000000.0;
const int MAXN = 900;///total nodes
```

```
const int MAXM = 100000;///total edges

int N,edges;
int last[MAXN],preve[MAXM],head[MAXM];
double Cap[MAXM],Flow[MAXM];
int dist[MAXN];
int nextEdge[MAXN];///used for keeping track of next edge of ith node
int ara[MAXM];

queue<int> Q;
#define pb push_back

void init(int N)
{
   edges=0;
   memset(last,-1,sizeof(last));
}
#define EPS 0

//cap=capacity of edges , flow = initial flow
vector<int>vc[3];
inline void addEdge(int u,int v,double cap,double flow,int bl)
{
   vc[bl].pb(edges);
   ara[edges]=u;
   head[edges]=v;
   preve[edges]=last[u];
   Cap[edges]=cap;
   Flow[edges]=flow;
   last[u]=edges++;
   vc[bl].pb(edges);
   head[edges]=u;
   preve[edges]=last[v];
   Cap[edges]=0;
   Flow[edges]=0;
   last[v]=edges++;
}

inline bool dinicBfs(int S,int E,int N)
{
   int from=S,to;
   double cap,flow;
   memset(dist,0,sizeof(int)*N);
   dist[from]=1;
   while(!Q.empty())
      Q.pop();
   Q.push(from);
   while(!Q.empty())
   {
      from=Q.front();
      Q.pop();
      for(int e=last[from]; e>=0; e=preve[e])
      {
         to=head[e];
         cap=Cap[e];
         flow=Flow[e];
         if(!dist[to] && (cap-flow)>EPS)
         {
            dist[to]=dist[from]+1;
            Q.push(to);
            ///Important
            if(to==E)
```

```
                return true;
            ///Need to be sure
          }
        }
    }
    return (dist[E]!=0);
}

inline double dfs(int from,double minEdge,int E)
{
    if(minEdge<=EPS)
        return 0;
    if(from==E)
        return minEdge;
    int to,e;
    double cap,flow,ret;
    for(; nextEdge[from]>=0; nextEdge[from]=preve[e])
    {
        e=nextEdge[from];
        to=head[e];
        cap=Cap[e];
        flow=Flow[e];
        if(dist[to]!=dist[from]+1)
            continue;
        ret=dfs(to,min(minEdge,(double)(cap-flow)),E);
        if(ret>EPS)
        {
            Flow[e]+=ret;
            Flow[e^1]-=ret;
            return ret;
        }
    }
    return 0;
}

double dinicUpdate(int S,int E)
{
    double flow=0;
    while(double minEdge = dfs(S,INF,E))
    {
        if(minEdge==0)
            break;
        flow+=minEdge;
    }
    return flow;
}

double maxFlow(int S,int E,int N)
{
    double totFlow=0;
    while(dinicBfs(S,E,N))
    {
        for(int i=0; i<=N; i++)
            nextEdge[i]=last[i];
        totFlow+=dinicUpdate(S,E);
    }
    return totFlow;
}
int deg[200];
vector<int>getflow(int n,int m,int source,int sink,double mid)
{
    for(int i=0; i<edges; i++)
```

```
{
   Flow[i]=0;
}
for(int i=0; i<vc[1].size(); i+=2)
{
   int h=ara[vc[1][i]];
   Cap[vc[1][i]]=(double)m+2.0*mid-(double)deg[h];
}
double rt=maxFlow(source,sink,n+5);
// cout<<rt<<endl;
int vis[200];
memset(vis,0,sizeof vis);
queue<int>qq;
vector<int>sc;
qq.push(source);
vis[source]=1;
sc.pb(0);
//  cout<<sc.size()<<endl;
while(!qq.empty())
{
   int from=qq.front();
   qq.pop();
   // cout<<from<<endl;
   if(from!=source)
      sc.pb(from);
   for(int e=last[from]; e>=0; e=preve[e])
   {
      int to=head[e];
      double cap=Cap[e];
      double flow=Flow[e];
      if(!vis[to]&&(cap-flow)>EPS)
      {
         vis[to]=1;
         qq.push(to);
      }
   }
  }
  return sc;

}
vector<int>ans;
void solve(int n,int m,int source,int sink)
{
   ans.clear();
   ans.pb(0);
   double lo=0,high=(double)m,ret=-1,f=1.0*(double)n*(n-1);
   while(f*(high-lo)>=1.0)
   {
      double mid=(lo+high)/2.0;
      vector<int>vc=getflow(n,m,source,sink,mid);
      if(vc.size()==1)
      {
         high=mid;
      }
      else
      {
         ans=vc;
         lo=mid;
      }
   }
}
int main()
```

```
{
   int n,m,ck=0;
   while(scanf("%d%d",&n,&m)==2)
   {
      int source=n+1;
      int sink=n+2;
      init(n+5);
      vc[0].clear();
      vc[1].clear();
      memset(deg,0,sizeof deg);
      for(int i=1; i<=m; i++)
      {
         int l,k;
         scanf("%d%d",&l,&k);
         addEdge(l,k,1,0.0,0);
         addEdge(k,l,1,0.0,0);
         deg[l]++;
         deg[k]++;
      }
      for(int i=1; i<=n; i++)
      {
         addEdge(source,i,(double)m,0.0,0);
      }
      for(int i=1; i<=n; i++)
         addEdge(i,sink,0.0,0.0,1);
      //  assert(edges<MAXM);
      solve(n,m,source,sink);
      if(ans.size()==1)
         ans.pb(1);
      if(ck)
         printf("\n");
         ck=1;
         printf("%d\n",ans.size()-1);
         sort(ans.begin(),ans.end());

         for(int i=1; i<ans.size(); i++)
         {
            printf("%d\n",ans[i]);
         }
   }
}
```

5. Unique Min Cut

```
//Dinic-Max Flow full code
int col[MAXN];
void dfs1(int now) {
   if(col[now]) return;
   //print1(now);
   col[now]=true;
   for(int e=Last[now]; e>=0; e=Prev[e]) {
      if(e&1) continue; //backward edge
      if(Cap[e]>Flow[e])
         dfs1(Head[e]);
   }
}
void dfs2(int now) {
   if(col[now]) return;
   //print1(now);
   col[now]=true;
   for(int e=Last[now]; e>=0; e=Prev[e]) {
      if((e&1)==0) continue; //forward edge
      if(Cap[e^1]>Flow[e^1])
```

```
            dfs2(Head[e]);
        }
    }
int main() {
    int n,m,a,b;
    while(cin>>n>>m>>a>>b &&(n||m||a||b)) {
        init(n+10);
        int u,v,w;
        int i;
        for(i=1; i<=m; i++) {
            scanf("%d %d %d",&u,&v,&w);
            addEdge(u,v,w,0);
            addEdge(v,u,w,0);
        }
        int augmentpath=maxFlow(a,b,n+3);
        mem(col,false);
        dfs1(a);
        dfs2(b);
        for(i=1; i<=n; i++)
            if(!col[i])
                break;
        if(i>n) print1("UNIQUE");
        else print1("AMBIGUOUS");
    }
    return 0;
}
```

6. BPM (Knuh Algorithm)

```
#include <bits/stdc++.h>
using namespace std;
int lefto[150],righto[150],visited[150];
vector<int>vc[150];
bool khun(int u)
{
    for(int i=0; i<vc[u].size(); i++)
    {
        int v=vc[u][i];
        if(visited[v]==1)
            continue;
        visited[v]=1;
        if(righto[v]==-1||khun(righto[v]))
        {
            lefto[u]=v;
            righto[v]=u;
            return true;
        }
    }

    return false;
}
int main()
{
    int i,j,k,l,m,n,test,u=1;
    scanf("%d",&test);
    while(test--)
    {
        scanf("%d",&n);
        for(i=1; i<=n; i++)
        {
            scanf("%d",&lefto[i]);
        }
        scanf("%d",&m);
```

```
    for(i=1; i<=m; i++)
    {
       scanf("%d",&righto[i]);
    }
    for(i=1; i<=n; i++)
    {
       for(j=1; j<=m; j++)
       {
          if(righto[j]%lefto[i]==0)
          {
             vc[i].push_back(j);
          }

       }
    }
    int cnt=0;
    memset(lefto,-1,sizeof lefto);
    memset(righto,-1,sizeof righto);
    for(i=1; i<=n; i++)
    {
       memset(visited,0,sizeof visited);
       if(khun(i))
          cnt++;

    }
    for(i=1; i<=n; i++)
       vc[i].clear();
    printf("Case %d: %d\n",u++,cnt);
  }
}
```

7. Bridge Tree

```
#define pb push_back
struct node
{
   int fir,sec;
   node() {}
   node(int _fir,int _sec)
   {
      fir=_fir;
      sec=_sec;
   }
};
vector<node>vc[200000];

int col[200000],distime[200000],height[200000],id,c,in[5005];

stack<int>st;
void  dfs_point(int u,int par)
{
   st.push(u);
   int v,child=0;
   distime[u]=height[u]=++id;
   for(int i=0; i<vc[u].size(); i++)
   {
      v=vc[u][i].fir;

      if(vc[u][i].sec==par)
         continue;
      child++;
      if(!distime[v])
      {

         dfs_point(v,vc[u][i].sec);
```

```
        height[u]=min(height[u],height[v]);
        /* if(distime[u]<height[v])
         {
            c++;
           while(st.size()){
            col[st.top()]=c;
            st.pop();
           }
         }*/
      }
      else if(distime[v]<distime[u])
      {
        height[u]=min(distime[v],height[u]);
      }
   }
   if(distime[u]==height[u])
   {
      c++;
      while(1)
      {
        col[st.top()]=c;
        if(st.top()==u)
           break;
        st.pop();
      }
      st.pop();
   }
}
int main()
{
   int i,j,k,l,m,n;
   vector<int>rev[5005];
   scanf("%d%d",&n,&m);
   for(i=1; i<=m; i++)
   {
      scanf("%d%d",&l,&k);
      assert(l!=k);
      vc[l].pb(node(k,i));
      vc[k].pb(node(l,i));
   }
   for(i=1; i<=n; i++)
   {
      if(distime[i])
         continue;
      dfs_point(i,-1);
   }
   c++;
   assert(st.size()==0);
   while(st.size())
   {
      col[st.top()]=c;
      st.pop();
   }
   for(int i=1; i<=n; i++)
   {
      for(int j=0; j<vc[i].size(); j++)
      {
         int w=vc[i][j].fir;
         if(col[w]==col[i])
            continue;
         in[col[i]]++;
      }
   }
   int ans=0;
   for(i=1; i<=c; i++)
   {
```

```
    if(in[i]==1)
      ans++;
  }
  printf("%d\n",(ans+1)/2);
}
```

8. HopcroftKarp (BPM Unweighted)

```
//Esqrt(V) Complexity
//0 Based
//Edge from set a to set b
const int MAXN1 = 50010;        //nodes in set a
const int MAXN2 = 50010;        //nodes in set b
const int MAXM = 150010;         //number of edges

int n1, n2, edges, last[MAXN1], prev[MAXM], head[MAXM];
int matching[MAXN2], dist[MAXN1], Q[MAXN1];
bool used[MAXN1], vis[MAXN1]; //vis is cleared in each dfs
// n1 = number of nodes in set a, n2 = number of nodes in set b
void init(int _n1, int _n2) {
  n1 = _n1;
  n2 = _n2;
  edges = 0;
  fill(last, last + n1, -1);
}
void addEdge(int u, int v) {
  head[edges] = v;
  prev[edges] = last[u];
  last[u] = edges++;
}
void bfs() {
  fill(dist, dist + n1, -1);
  int sizeQ = 0;
  for (int u = 0; u < n1; ++u) {
    if (!used[u]) {
      Q[sizeQ++] = u;
      dist[u] = 0;
    }
  }
  for (int i = 0; i < sizeQ; i++) {
    int u1 = Q[i];
    for (int e = last[u1]; e >= 0; e = prev[e]) {
      int u2 = matching[head[e]];
      if (u2 >= 0 && dist[u2] < 0) {
        dist[u2] = dist[u1] + 1;
        Q[sizeQ++] = u2;
      }
    }
  }
}
bool dfs(int u1) {
  vis[u1] = true;
  for (int e = last[u1]; e >= 0; e = prev[e]) {
    int v = head[e];
    int u2 = matching[v];
    if (u2 < 0 || (!vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u2))) {
      matching[v] = u1;
      used[u1] = true;
      return true;
    }
  }
  return false;
}
```

```
int augmentPath() {
   bfs();
   fill(vis, vis + n1, false);
   int f = 0;
   for (int u = 0; u < n1; ++u)
      if (!used[u] && dfs(u))
         ++f;
   return f;
}
int maxMatching() {
   fill(used, used + n1, false);
   fill(matching, matching + n2, -1);
   for (int res = 0;;) {
      int f = augmentPath();
      if (!f)
         return res;
      res += f;
   }
}
```

9. Hungarian

```
#include<bits/stdc++.h>
using namespace std;

typedef int ll;
const int N=1023, INF=1e9, NPOS=-1;
struct Matrix
{
   int n;
   ll a[N][N];
};
```

```
struct KM:Matrix
{
   vector<ll> hl, hr, slk;
   vector<int> fl, fr, vl, vr, pre;
   deque<int> q;
   int check(int i)
   {
      if(vl[i]=1, fl[i]!=-1) return q.push_back(fl[i]), vr[fl[i]] = 1;
      while(i!=-1) swap(i, fr[fl[i]=pre[i]]);
      return 0;
   }

   void bfs(int s)
   {
      slk.assign(n, INF), vl.assign(n, 0), vr=vl, q.assign(vr[s]=1, s);
      for(ll d; ;)
      {
         for(; !q.empty(); q.pop_front())
            for(int i=0, j=q.front(); i<n; i++)
               if(d=hl[i]+hr[j]-a[i][j], !vl[i]&&d<=slk[i])
                  if(pre[i]=j, d) slk[i]=d;
                  else if(!check(i)) return;
         d=INF;
         for(int i=0; i<n; i++)
            if(!vl[i] && d>slk[i]) d = slk[i];
         for(int i = 0; i<n; i++)
         {
            if(vl[i]) hl[i]+=d;
            else slk[i]-=d;
            if(vr[i]) hr[i]-=d;
         }
```

```
        for(int i = 0; i<n; i++)
            if(!vl[i]&&!slk[i]&&!check(i)) return;
        }
    }
    void ask()
    {
        fl.assign(n, -1), fr=fl, hl.assign(n, 0), pre=hr=hl;
        for(int i = 0; i<n; i++) hl[i] = *max_element(a[i], a[i]+n);
        for(int j=0; j<n; j++) bfs(j);
    }
};

int main()
{
    int n, m, k;
    scanf("%d %d %d", &m, &n, &k);
    KM solv;
    solv.n = max(n, m);
    for(int i = 0; i<k; i++)
    {
        int u, v, w;
        scanf("%d %d %d", &u, &v, &w);
        u--, v--;
        solv.a[u][v] = w;

    }
    solv.ask();
    int e = 0, ans = 0;
    for(int i = 0; i<n; i++)
        if(solv.a[i][solv.fl[i]]) e++, ans += solv.a[i][solv.fl[i]];
    printf("%d\n%d\n", ans, e);
    for(int i = 0; i<n; i++)
    {
        if(solv.a[i][solv.fl[i]]) printf("%d %d\n", i+1, 1+solv.fl[i]);
    }
    return 0;
}
```

10. Blossom Algorithm

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <vector>
using namespace std;

const int NMax = 230;
int Next[NMax];
int spouse[NMax];
int belong[NMax];

int findb(int a)
{
    return belong[a]==a?a:belong[a]=findb(belong[a]);
}

void together(int a, int b)
{
    a = findb(a), b = findb(b);
    if(a != b) belong[a] = b;
}

vector<int> E[NMax];
```

```
int N;
int Q[NMax], bot;
int mark[NMax];
int visited[NMax];
int findLCA(int x, int y)
{
   static int t = 0;
   t++;
   while(1)
   {
      if(x!=-1)
      {
         x = findb(x);
         if(visited[x]==t) return x;
         visited[x]=t;
         if(spouse[x]!=-1) x = Next[spouse[x]];
         else x = -1;
      }
      swap(x, y);
   }
}

void goup(int a, int p)
{
   while(a != p)
   {
      int b = spouse[a], c = Next[b];
      if(findb(c) != p) Next[c] = b;
      if(mark[b]==2) mark[Q[bot++]=b]=1;
      if(mark[c]==2) mark[Q[bot++]=c]=1;
      together(a, b);
      together(b, c);
      a = c;
   }
}
void findaugment(int s)
{
   for(int i = 0; i<N; i++) Next[i]=-1, belong[i]=i, mark[i]=0, visited[i]=-1;
   Q[0]=s, bot=1, mark[s] = 1;
   for(int head=0; spouse[s]==-1 && head<bot; head++)
   {
      int x=Q[head];
      for(int i = 0; i<(int)E[x].size(); i++)
      {
         int y = E[x][i];
         if(spouse[x]!=y && findb(x)!=findb(y) && mark[y]!=2)
         {
            if(mark[y]==1)
            {
               int p = findLCA(x, y);
               if(findb(x) != p) Next[x] = y;
               if(findb(y) != p) Next[y] = x;
               goup(x, p);
               goup(y, p);
            }
            else if(spouse[y]==-1)
            {
               Next[y] = x;
               for(int j=y; j!=-1;)
               {
                  int k = Next[j];
```

```
                int l = spouse[k];
                spouse[j] = k;
                spouse[k] = j;
                j = l;
              }
              break;
            }
            else
            {
              Next[y] = x;
              mark[Q[bot++] = spouse[y]] = 1;
              mark[y] = 2;
            }
          }
        }
      }
}

int main()
{
  int n;
  scanf("%d", &n);
  N = n;
  for(int i = 0; ; i++)
  {
    int u, v;
    int x = scanf("%d %d", &u, &v);
    if(x == EOF) break;
    u--, v--;
    E[u].push_back(v);
    E[v].push_back(u);
  }
  memset(spouse, -1, sizeof spouse);
  for(int i = 0; i<n; i++)
  {
    if(spouse[i] == -1) findaugment(i);
  }
  int ans = 0;
  for(int i = 0; i<n; i++)
  {
    if(spouse[i] != -1) ans++;
  }


  printf("%d\n", ans);
  for(int i = 0; i<n; i++)
  {
    if(spouse[i]>-1 && spouse[i]>i)
    {
      printf("%d %d\n", i+1, spouse[i]+1);
    }
  }
}
```

11. BronKerbosch (Maximum clique)

```
/* Find Maximum clique in a graph .
Edges are stored using bit.
BronKerbosch(0,(1LL<<node) - 1 ,0)  */
long long n, edges[50], fans ;
void BronKerbosch(long long r,long long p,long long x) {
  if(p == 0 && x == 0) {
    fans = max(fans,(long long)__builtin_popcountll(r));
```

```
      return ;
   }
   int u = 0 ;
   while(!((1LL<<u) & (p|x)))
      u++;
   for(int v=0 ; v<n ; v++) {
      if(((1LL<<v)&p) && !((1LL<<v) & edges[u])) {
         BronKerbosch(r|(1LL<<v),p&edges[v],x&edges[v]);
         p -= (1LL<<v) ;
         x |= (1LL<<v) ;
      }
   }
}
```

12. Bellman Ford

```
//complexity VE
#define SIZE 1010
#define INF 2000000000
vector<int> adj[SIZE],cost[SIZE];
//0 based
bool BellmanFord(int source,int nodes) { //returns true if it has negative
cycle
   vector<int>dist;
   int i,j,k,w,v;
   for(i=0; i<=nodes; i++) { //distance from source
      dist.push_back(INF);
   }
   dist[source]=0;
   for(i=1; i<=nodes-1; i++) {
      for(j=1; j<=nodes; j++)
         for(k=0; k<adj[j].size(); k++) {
            v=adj[j][k];
            w=cost[j][k];
            dist[v]=min(dist[v],dist[j]+w);
         }
   }
   for(i=1; i<=nodes; i++)
      for(j=0; j<adj[i].size(); j++) {
         v=adj[i][j];
         w=cost[i][j];
         if(dist[v]>dist[i]+w) return true;
      }
   return false;
}
```

13. Directed MST

```
#define MAX_VERTEX 100100
#define INF 100100000
struct Edge {
   int u,v,w,ind;
   Edge(int u=0,int v=0,int w=0) {
      this->u = u;
      this->v = v;
      this->w = w;
   }
   bool operator < (const Edge &b)
   const {
      return w<b.w;
   }
};
int nV,nE; //nV -> Number of Vertex.
vector<Edge> Edges[MAX_VERTEX]; //Adjecency List.
```

```
//Edge u->v inserted into list of v.
vector<Edge> EdgeList; //All edges. Used
//if Path or Used Edges Required.
vector<int>adj[MAX_VERTEX]; // to check the
//graph connectivity.
int par[MAX_VERTEX],color[MAX_VERTEX];
int W[MAX_VERTEX],toUse[MAX_VERTEX];
bool used[MAX_VERTEX+100];
int vertexEdge[MAX_VERTEX];
vector<int>choosed;
int DMST(int nodes,int root,vector<Edge> Edges[]) {
   int i,j,t,u,v;
   Edges[root].clear();
   for(i=0; i<nodes; i++) {
      par[i] = i;
      sort(Edges[i].begin(),Edges[i].end());
   }
   bool cycle_found = true;
   while(cycle_found) {
      cycle_found = false;
      memset(color,0,sizeof color);
      color[root] = -1;
      for(i=0,t=1; i<nodes; i++,t++) {
         u = par[i];
         if(color[u]) continue;
         for(v=u; !color[v]; v=par[Edges[v][0].u]) {
            color[v] = t;
            choosed.push_back(Edges[v][0].ind);
         }
         if(color[v] != t) continue;
         cycle_found = true;
         int sum = 0, super = v;
         for( ; color[v]==t; v=par[Edges[v][0].u]) {
            color[v]++;
            sum+= Edges[v][0].w;
         }
         for(j=0; j<nodes; j++) W[j] = INF;
         for(; color[v]==t+1; v=par[Edges[v][0].u]) {
            color[v]--;
            for(j = 1; j<Edges[v].size(); j++) {
               int w = Edges[v][j].w+
                     sum-Edges[v][0].w;
               if(w<W[Edges[v][j].u]) {
                  W[Edges[v][j].u] = w;
                  toUse[Edges[v][j].u]=Edges[v][j].ind;
               }
            }
            par[v] = super;
         }
         Edges[super].clear();
         for(j=0; j<nodes; j++)
            if(par[j] != par[par[j]])
               par[j] = par[par[j]];
         for(j=0; j<nodes; j++)
            if(W[j]<INF && par[j]!= super) {
               Edge e = Edge(j,super,W[j]);
               e.ind = toUse[j];
               Edges[super].push_back(e);
            }
         sort(Edges[super].begin(),Edges[super].end());
         for(j=0; j<Edges[super].size(); j++) {
            Edge e=Edges[super][j];
```

```
        }
      }
   }
//cout<<"In outside of Loop:"<<endl;
   int sum = 0;
   for(i=0; i<nodes; i++)
      if(i!=root && par[i]==i) {
         sum += Edges[i][0].w;
// i'th node's zero'th edge contains the
//minimum cost after DMST algo.
      }
   return sum;
}//End Of DMST Function....
int isPossible() {
   int i,j,u,v;
   for(i=0; i<nV; i++) {
      for(j=0; j<Edges[i].size(); j++) {
         adj[Edges[i][j].u].push_back(Edges[i][j].v);
      }
   }
   queue<int>Q;
   Q.push(0);
   memset(color,0,sizeof color);
   color[0] = 1;
   while(!Q.empty()) {
      //BFS to check graph Connectivity.
      u = Q.front();
      Q.pop();
      for(i=0; i<adj[u].size(); i++) {
         v = adj[u][i];
         if(color[v]) continue;
         color[v] = 1;
         Q.push(v);
      }
   }
   for(i=0; i<nV; i++) if(!color[i]) return -1;
   return DMST(nV,0,Edges);
}
int main() {
   int i,j,test,Case=1;
   Edge e;
   test = 1;
   while(test--) {
      scanf("%d %d",&nV,&nE);
      for(i=0; i<nE; i++) {
         scanf("%d %d %d",&e.u,&e.v,&e.w);
         e.u--;
         e.v--;
         e.ind = i;
         Edges[e.v].push_back(e);
         EdgeList.push_back(e);
      }
      memset(used,0,sizeof used);
      int res = isPossible();
      if(res == -1) printf("-1\n");
      else {
         memset(used,0,sizeof used);
         memset(color,0,sizeof color);
         for(i=choosed.size()-1; i>=0; i--) {
            Edge e = EdgeList[choosed[i]];
            if(color[e.v]) continue;
            color[e.v] = 1;
```

```
            used[choosed[i]] = true;
        }
        printf("%d\n",res);
        if(res) {
            for(i=0; i<nE; i++)
                if(used[i] && EdgeList[i].w)
                    printf("%d ",i+1);
            printf("\n");
        }
    }
  }
  return 0;
}
```

14. Dominator Tree

```
// note: Here root is 1
const int maxn = 200900;
vector<int> graph[maxn];
vector<int> tree[maxn], rg[maxn], bucket[maxn];
int sdom[maxn], par[maxn], dom[maxn], dsu[maxn], label[maxn];
int arr[maxn], rev[maxn], T ;
void dini(int node) {
  T = 0;
  for (int i = 0; i <= node ; i++) {
    sdom[i] = par[i] = dom[i] = dsu[i] = label[i] = 0;
    arr[i] = rev[i] = 0;
    graph[i].clear();
    tree[i].clear();
    rg[i].clear();
    bucket[i].clear();
  }
}
int Find(int u, int x = 0) {
  if (u == dsu[u])
    return x ? -1 : u;
  int v = Find(dsu[u], x + 1);
  if (v < 0)
    return u;
  if (sdom[label[dsu[u]]] < sdom[label[u]])
    label[u] = label[dsu[u]];
  dsu[u] = v;
  return x ? v : label[u];
}
void Union(int u, int v) { //Add an edge u-->v
  dsu[v] = u;
}
void dfs0(int u) {
  T++;
  arr[u] = T;
  rev[T] = u;
  label[T] = T;
  sdom[T] = T;
  dsu[T] = T;
  for (int i = 0; i < graph[u].size() ; i++) {
    int w = graph[u][i];
    if (!arr[w])
      dfs0(w), par[arr[w]] = arr[u];
    rg[arr[w]].push_back(arr[u]);
  }
}
void BuildTree() {
  //Build Dominator tree
```

```
    dfs0(1);
    int n = T;
    for (int i = n; i >= 1; i--) {
        for (int j = 0; j < rg[i].size() ; j++)
            sdom[i] = min(sdom[i], sdom[Find(rg[i][j])]);
        if (i > 1)
            bucket[sdom[i]].push_back(i);
        for (int j = 0; j < bucket[i].size() ; j++) {
            int w = bucket[i][j];
            int v = Find(w);
            if (sdom[v] == sdom[w])
                dom[w] = sdom[w];
            else
                dom[w] = v;
        }
        if (i > 1)
            Union(par[i], i);
    }
    for (int i = 2; i <= n; i++) {
        if (dom[i] != sdom[i])
            dom[i] = dom[dom[i]];
        tree[rev[i]].push_back(rev[dom[i]]);
        tree[rev[dom[i]]].push_back(rev[i]);
    }
}
```

15. Euler Path Print

```
int is[sz] = {0} ;
vector<int>path ;
void go(int u) {
    while(is[u]<adj[u].size())
        go(adj[u][is[u]++]);
    ans.push_back(u);
}
/// In ans vector path will be saved in reverse order
```

16. Articulation Points (or Cut Vertices)

```
const int sz = 1007 ;
vector<int>graph[sz] ;
int low[sz], disc[sz], tme, ans, compo, component[sz] ;
bool isPoint[sz] ;
void reset() {
    for(int i=0 ; i<sz ; i++) {
        graph[i].clear();
        low[i] = -1 ;
        disc[i] = -1 ;
        component[i] = -1 ;
        isPoint[i] = 0 ;
    }
    tme = 0 ;
    ans = 0 ;
    compo = 0 ;
}

void tarjan(int u,int p) {
    low[u] = disc[u] = ++tme ;
    int v, cont = 1, childern = 0 ;
    for(int i=0 ; i<graph[u].size() ; i++) {
        v = graph[u][i] ;
        if(v == p || v == u)
```

```
            continue;
        if(disc[v] == -1) {
            childern++;
            tarjan(v,u);
            low[u] = min(low[u],low[v]);
            if(p == -1 && childern>1)
                cont++, isPoint[u] = 1 ;
            if(p != -1 && low[v]>=disc[u])
                cont++, isPoint[u] = 1 ;
        } else
            low[u] = min(low[u],disc[v]);
    }
    component[u] = cont ;
}
```

17. Articulation Bridge

```
#define lim         1005
//in multiple edge bridge won't work
int tim[lim],low[lim];
int timer;
vector<int> adj[lim]; //only adj should be cleared
struct edge {
    int u, v;
};
vector<edge> bridge;//the ans(should be cleared)
void dfs(int u,int par) { //par=-1 dhore call dite hobe(root ar parent nai)
    tim[u] = low[u] = ++timer;

    for(int i = 0 ; i<adj[u].size() ; i++) {
        int v = adj[u][i];
        if(v==par) continue;
        if(!tim[v]) {
            dfs(v,u);
            low[u] = min(low[u],low[v]);
            if(low[v]>tim[u]) { //attention greater equals for bridge and
articulation point
                edge tem;
                tem.u=u;
                tem.v=v;
                bridge.push_back(tem);
            }
        } else { //determining back edge
            low[u] = min(low[u],tim[v]);
        }
    }
    return;
}

//sometimes change needed here
void articulation_bridge(int n) {
    memset(tim,0,sizeof tim);
    timer=0;
    for(int i=1; i<=n; i++)
        if(!tim[i])
            dfs(i,-1);
}
```

18. BCC

```
//1 Based
//no problem in multiple edge and self loop
int tim[lim],low[lim];
int timer;
```

```
vector<int> adj[lim]; //only adj should be cleared
stack<pair<int,int> >S;
pair<int,int> ed[2*lim];//because one edge can be part of two BCC

void calc_bcc(int u, int v) {
   int i, j, uu, vv, cur;
   pair<int,int> now;
   int tot=0;
   while(!S.empty()) {
      now = S.top();
      S.pop();
      uu = now.first, vv = now.second;
      ed[tot++] = make_pair(uu, vv);
      if(u==uu && v==vv) break;
      if(u==vv && v==uu) break;
   }
   if(tot<=1) return;
   puts("");
   for(int i=0; i<tot; i++) {
      cout<<ed[i].first<<" "<<ed[i].second<<" ";
   }
   cout<<endl;
   //doing according to problem
   return;
}
void bcc(int u,int par) {
// par=-1 dhore call dite hobe(root ar parent nai)
   tim[u] = low[u] = ++timer;
   for(int i = 0 ; i<adj[u].size() ; i++) {
      int v = adj[u][i];
      if(v==par) continue;
      if(tim[v]==0) {
         S.push(make_pair(u, v));
         bcc(v,u);
         low[u] = min(low[u],low[v]);
         if(low[v]>=tim[u]) {
            cout<<"cheak : "<<u<<' '<<v<<endl;
            calc_bcc(u, v);
         }
      }
      else if(tim[v] < tim[u]) {
         low[u] = min(low[u],tim[v]);
         S.push(make_pair(u, v));
      }
   }
   return;
}
void BCC(int n) {
   timer=0;
   memset(tim,0,sizeof tim);
   int i;
   for(i = 1; i <= n; i++)
      if(!tim[i])
         bcc(i,-1);
}
void add(int ina,int inb) {
   adj[ina].push_back(inb);
   adj[inb].push_back(ina);
}
int main() {
   int n,m,u,v;
   cin>>n>>m;
```

```
    while(m--) {
        cin>>u>>v;
        add(u,v);
    }
    BCC(n);
}
```

19. SCC

```
/**
*   Tarjan Algorithm Starts From here
*   Call reset() before calling Tarjan()
*   Graph Directed
*/
const int MX = 1000;  /// Maximum Node

int low[MX+7], dis[MX+7];
int belong[MX+7];  /// for storing SCC no. of each node
int tym;  /// timer for DFS Tree
int SCC;  /// For Counting SCC no.
vector<int> adj[MX+7];
stack<int> stk;  /// stack for tracking nodes of same SCC

void reset(){
    tym = SCC = 0;
    for( int i = 0; i<=MX; i++ ){
        low[i] = belong[i] = -1;
        adj[i].clear();
    }
    while( !stk.empty() )
        stk.pop();
}

void tarjan(int u){
    int v, i;
    low[u] = dis[u] = ++tym;
    stk.push(u);

    for( i = 0; i<adj[u].size(); i++ ){
        v = adj[u][i];
        if( low[v] == -1 ){ /// Tree Edge
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else if( belong[v] == -1 ){   /// Back Edge
            low[u] = min(low[u], dis[v]);
        }
    }

    if( low[u] == dis[u] ){
        v = stk.top();
        while( v != u ){
            belong[v] = SCC;
            stk.pop();
            v = stk.top();
        }
        belong[v] = SCC++;
        stk.pop();
    }
}

int main(){
    //freopen("E:\\00.txt", "r", stdin);
```

```
   return 0;
}
```

20. 2-SAT (Jubair)

```
#include "bits/stdc++.h"
using namespace std;

#define SZ  2*8010
#define SZ1  20010

struct data
{
   int x, y;
};
data val[SZ1];

vector <int> adj[SZ];
int col[SZ], low[SZ], tim[SZ], timer;
int group_id[SZ], compo;
bool logic[SZ];
stack <int> S;

inline void inp( int &n )
{
      n=0;
      int ch=getchar_unlocked();int sign=1;
      while( ch < '0' || ch > '9' ){if(ch=='-')sign=-1; ch=getchar_unlocked();}

      while(  ch >= '0' && ch <= '9' )
            n = (n<<3)+(n<<1) + ch-'0', ch=getchar_unlocked();
      n=n*sign;
}
```

```
void SCC(int u)
{
   int i, v, tem, k;

   col[u] = 1;
   low[u] = tim[u] = timer++;
   S.push(u);

   k = (int)adj[u].size();

   for(i=0; i<k; i++)
   {
      v = adj[u][i];
      if(col[v] == 1)
         low[u]= min(low[u], tim[v]);
      else if(col[v] == 0)
      {
         SCC(v);
         low[u] = min(low[u], low[v]);
      }
   }


   if(low[u] == tim[u])
   {
      do
      {
         tem = S.top();
         S.pop();
         group_id[tem] = compo;
         col[tem] = 2;
      }
      while(tem != u);
```

```
      compo++;
   }
}

void tarjan_SCC(int m)
{
   int i;

   for(i=0; i<=2*m+2; i++)
   {
      col[i] = 0;
   }

   timer = compo = 0;
   while(!S.empty()) S.pop();

   for(i=2; i<=2*m+1; i++)
      if(col[i] == 0)
         SCC(i);
}
void implication_graph(int n)
{
   int i, j, k, a, b, aprime, bprime;

   for(i=0; i<n; i++)
   {
      j = val[i].x;
      k = val[i].y;
      if(j > 0)
      {
         a = 2 * j;
         aprime = 2 * j + 1;
      }
      else
      {
         a = 2 * (-j) + 1;
         aprime = 2 * (-j);
      }

      if(k > 0)
      {
         b = 2 * k;
         bprime = 2 * k + 1;
      }
      else
      {
         b = 2 * (-k) + 1;
         bprime = 2 * (-k);
      }
      adj[aprime].push_back(b);
      adj[bprime].push_back(a);
   }

}
bool two_SAT(int m)
{
   int i;

   for(i=2; i<=2*m; i+=2)
   {
      if(group_id[i] == group_id[i+1])
         return false;
      else if(group_id[i] > group_id[i+1])
         logic[i/2] = false;
      else
         logic[i/2] = true;
```

```
    }

    return true;
}
int main()
{
    int test=0, t, n, m, i, j, k;
    vector <int> v;

    inp(t);

    while(t--)
    {
        inp(n);
        inp(m);

        for(i=0; i<=2*m+2; i++)
            adj[i].clear();

        for(i=0; i<n; i++)
        {
            inp(val[i].x);
            inp(val[i].y);
        }

        implication_graph(n);

        tarjan_SCC(m);
        printf("Case %d: ",++test);

        if(two_SAT(m))
        {
            printf("Yes\n");
            v.clear();

            for(i=1; i<=m; i++)
                if(logic[i])
                    v.push_back(i);

            k = (int)v.size();
            printf("%d",k);

            for(i=0; i<k; i++)
                printf(" %d",v[i]);

            printf("\n");
        }
        else
            printf("No\n");
    }

    return 0;
}
```

21. Articulation Point (Jubair)

```
vector<int> adj[maxnode];
vector<int> point;
int distime[maxnode],height[maxnode],root,col[maxnode],id;

void dfs_point(int u,int par)
{
    int v,child=0;
    distime[u]=height[u]=id++;
    for(int i=0;i<adj[u].size();i++)
    {
```

```
        v=adj[u][i];
        if(v==par) continue;
        if(!distime[v])
        {
           child++;
           dfs_point(v,u);
           height[u]=min(height[u],height[v]);///height update.
           if(distime[u]<=height[v]&&u!=root&&!col[u])
           {
              point.psb(u);///find point.
              col[u]=true;
           }
        }
     if(distime[u]>distime[v]) height[u]=min(distime[v],height[u]);///back
edge.
    }
    if(u==root&&child>=2&&!col[u])
    {
       col[u]=true;
       point.psb(u);///root case.
    }
    return ;
}

int main()
{
    int test,n,m,u,v,res,cas=0;
    scanf("%d",&test);
    while(test--)
    {
       scanf("%d %d",&n,&m);
       for(int i=0;i<=n;i++) adj[i].clear();
       clr(distime,0);
       clr(height,0);
       clr(col,0);
       point.clear();
       for(int i=0;i<m;i++)
       {
          scanf("%d %d",&u,&v);
          adj[u].psb(v);
          adj[v].psb(u);
       }
       id=1;
       for(root=1;root<=n;root++)
       {
          if(!distime[root])
          {
             dfs_point(root,root);
          }
       }
       printf("Case %d: %d\n",++cas,point.size());
    }
    return 0;
}
```

22. Flow/BPM Notes

		Flow Algorithms	
	Name	**Complexity**	**Average Case**
1	Ford Fulkerson	VE^4	V^3
2	Dinic Maxflow	V^2E	V^3
3	Min cost using SPFA	VE^4	V^3
4	Hopcroft- carp	**E√V**	**E√V**
5	Hungarian	N^2M	N^2M
6	Non-weighted Blossoms	VE	VE
7	Weighted Blossom		
		N = number of rows, M = number of columns	
		Concepts	
	Name	**Description**	**Solution**
1	Vertex cover	Minimum number of vertex required to cover all edges	Equals to Matching for bipartite otherwise NP complete
2	Edge cover	Minimum number of edge required to cover all vertices	V-matching for all graphs
3	Minimum Independent path (IP)	Minimum number of disjoint paths to cover all vertices	V-matching for all graphs
4	Minimum path cover (MPC)	Minimum number of paths to cover all vertices	Convert it MIP problem by finding trasitive closure
5	Clique	A complete subgraph	
6	Maximal clique	A clique which cannot be expanded	
7	Maximum clique	A maximal clique with highest number of vertices	Make a reverse graph then answer = V - vertex cover
8	Closure	A directed subgraph with no outgoing edges outside the graph	

9	Max/min closure	A closure with max/min sum of weighted nodes	For max join source with positive nodes,sink with negative
			nodes and capacities are absolute value,infinite capacity
			between existing edges. For min,source & sink Is reversed
			Ans = sum of positive nodes - min cut (For max)
			Ans = sum of negative nodes + min cut (For min)
10	Interval graph	If the nodes can be defined by intervals, and edges are built	Can be solved without flow in nlogn complexity
		based on interval overlap	
11	Perfect matching	Everynode can be matched	
12	Minimum Dominating set	Minimum number of vertex to cover all vertices	NP-complete
13	Set cover	Minimum number of set to cover all elements	NP-complete
14	Hitting set	Minimum number of element to cover all sets	NP-complete
15	Minimum weighted matching	Maximum matching with Minimum cost	Adding a extra column in self matching which is much
			greater than the rest but much smaller than infinity.
			Then apply hungarian algorithm
16	Minimum weighted IP	Minimum IP with minimum weighted	Convert it to Minimum weighted matching
	Sometimes answer = matching/2		
`	Normally binary Search is better than iteration in flow		

Data Structure

23. Magic STL

```
// Edit Stl
#include <bits/stdc++.h>
using namespace std ;

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace __gnu_pbds;

typedef tree<
double,
int,
less<double>,
rb_tree_tag,
tree_order_statistics_node_update> map_t; //create map tree

typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update> set_t; //create set tree

int main()
{
   map_t s ;
   set_t ss ;
   s.insert(make_pair(12, 1012));
   s.insert(make_pair(505, 1505));
   s.insert(make_pair(30, 1030));
   s.insert(make_pair(12,580));

   cout<<s.find_by_order(0)->sc<<endl ; // find by indx
   cout<<s.order_of_key(20)<<endl; // count less than 20 by first element
   ss.insert(1);
   ss.insert(4);
   ss.insert(10);
   ss.insert(5);

   cout<<*ss.find_by_order(1)<<endl ;
   cout<<ss.order_of_key(0)<<endl ; //count less than 5
   ss.erase(4); // erase by element

   return 0;
}
```

24. BIT

```
int tree[Max] ;
void update(int idx,int x) {
   while(idx<=Max) {
      tree[idx] += x ;
      idx += (idx & (-idx));
   }
}
ll qury(int idx) {
   ll sum = 0 ;
```

```
    while(idx>0) {
        sum+=tree[idx];
        idx -= (idx&(-idx));
    }
    return sum ;
}
ll readsingle(int idx) {
    ll sum = tree[idx] ;
    if(idx>0) {
        int z = idx - (idx&(-idx)) ;
        idx--;
        while(z != idx) {
            sum -= tree[idx] ;
            idx -= (idx & (-idx));
        }
    }
    return sum ;
}
```

25. Build BST

```
struct Build_BST {
// Element of array must be a permutation of 1 to n
    set<int>st ;
    set<int>::iterator it ;
    vector<vector <int> >graph ;
    pair<int,int> isfree[sz] ;
    int arr[sz], ln ;
    Build_BST(int ln):ln(ln),graph(n+7) {
        for(int i=0 ; i<sz ; i++)
            isfree[i] = make_pair(-1,-1);
    }
    void add(int u) {
        int v ;
        it = st.lower_bound(u);
        if(it != st.end()) {
            v = *it ;
            if(isfree[v].first == -1) {
                isfree[v].first = 1 ;
                graph[v].push_back(u);
            }
        }
        it--;
        if(it != st.begin()) {
            v = *it ;
            if(isfree[v].second == -1) {
                isfree[v].second = 1 ;
                graph[v].push_back(u);
            }
        }
        st.insert(u);
    }
    void build() {
        st.insert(-1);
        st.insert(arr[0]);
```

```
    for(int i=1 ; i<ln ; i++)
        add(arr[i]);
  }
};
```

26. Persistent Segment Tree

```
struct data {
  int l, r, val;
  data() {
    l = r = val = 0;
  }
  data(int _l, int _r, int _val) {
    l = _l, r = _r, val = _val;
  }
} node[10*MXN+7];  /// node indexing from 1

int tree[MXN+7], cnt;
int build(int cur, int base, int top) {
  if( base==top ) {
    node[cur] = data(0, 0, 0);
    return 0;
  }
  int left, right, mid;
  node[cur].l = left = ++cnt;
  node[cur].r = right = ++cnt;
  mid = ( base+top )/2;
  node[cur].val = build(left, base, mid);
  node[cur].val += build(right, mid+1, top);
  return node[cur].val;
}
void upgrade(int pre, int cur, int base, int top, int pos, int v) {
  if( base==top ) {
    node[cur].val += v;
    return;
  }
  int left, right, mid;
  mid = ( base+top )/2;
  if( pos<=mid ) {
    node[cur].r = node[pre].r;
    node[cur].l = ++cnt;
    upgrade(node[pre].l, node[cur].l, base, mid, pos, v);
  } else {
    node[cur].l = node[pre].l;
    node[cur].r = ++cnt;
    upgrade(node[pre].r, node[cur].r, mid+1, top, pos, v);
  }
  node[cur].val = node[ node[cur].l ].val + node[ node[cur].r ].val;
}
int query(int pre, int cur, int base, int top, int pos) {
  if( base == top )   return base;
  int ele = node[ node[cur].l ].val - node[ node[pre].l ].val;
  int mid = ( base+top )/2;
  if( ele>=pos ) {
    return query(node[pre].l, node[cur].l, base, mid, pos);
  }
  return query(node[pre].r, node[cur].r, mid+1, top, pos-ele);
}
int arr[MXN+7];
/**   Problem: K-th Number in a Range. Assume all numbers between 1
to n and distinct    */
int main() {
```

```
//    freopen("E:\\00.txt", "r", stdin);
    int n, m, l, r, pos, res, i, j, k;
    cnt = 0;
    sf2(n, m);
    for( i = 1; i<=n; i++ ) {
        sf1(arr[i]);  /// arr[i] -> 1 to n
    }
    tree[0] = ++cnt;
    build(tree[1], 1, n);
    for( i = 1; i<=n; i++ ) {
        tree[i] = ++cnt;
        upgrade(tree[i-1], tree[i], 1, n, arr[i], 1);
    }
    for( i = 0; i<m; i++ ) {
        sf3(l, r, pos);
        res = query(tree[l-1], tree[r], 1, n, pos);
        pf("%d\n", res);
    }
    return 0;
}
```

27. Persistent Segment Tree with Lazy

```
#define pb push_back
#define MX 2800000
long long tree[MX];
int righto[MX],lefto[MX],nw=0,m;
int ara[100005];
int  root[MX];
long long lazy[MX];
int update(int node,int st,int en,int l,int r,long long d)
{
    if(st>r || en<l)
        return node;
    int psenode=++nw;
    tree[psenode]=tree[node];
    lazy[psenode]=lazy[node];
    lefto[psenode]=lefto[node];
    righto[psenode]=righto[node];
    if(st>=l && en<=r)
    {
        tree[psenode]=tree[psenode]+(en-st+1)*d;
        lazy[psenode]+=d;
        // if(st==3&&en==3)cout<<tree[psenode]<<endl;
        return psenode;
    }
    int mid=(st+en)/2;
    lefto[psenode]=update(lefto[node],st,mid,l,r,d);
    righto[psenode]=update(righto[node],mid+1,en,l,r,d);
    tree[psenode]=tree[lefto[psenode]]+tree[righto[psenode]]+(en-
st+1)*lazy[psenode];
    return psenode;
}
long long query(int node,int st,int en,int l,int r,long long carry)
{
    if(st>r || en<l)
        return 0;
    if(st>=l && en<=r)
    {
        //cout<<tree[node]<<" "<<carry<<endl;
        return tree[node]+(en-st+1)*carry;
    }
    int mid=(st+en)/2;
```

```
   return
query(lefto[node],st,mid,l,r,carry+lazy[node])+query(righto[node],mid+1,
en,l,r,carry+lazy[node]);

}
int build(int st,int en)
{
   int pse=++nw;
   if(st==en)
   {
      tree[pse]=ara[st];
      return pse;
   }
   int mid=(st+en)/2;
   lefto[pse]=build(st,mid);
   righto[pse]=build(mid+1,en);
   tree[pse]=tree[lefto[pse]]+tree[righto[pse]];
   return pse;

}
int main()
{
   int i,j,k,l,m,n,q,d;
   char str[7];
   scanf("%d%d",&n,&q);
   nw=0;
   memset(tree,0,sizeof tree);
   memset(lazy,0,sizeof lazy);
   for(i=1; i<=n; i++)
      scanf("%d",&ara[i]);
   root[0]=build(1,n);
   int t=0;
   for(i=1; i<=q; i++)
   {
      scanf("%s",str);
      if(str[0]=='C')
      {
         t++;
         scanf("%d%d%d",&l,&k,&d);
         root[t]=update(root[t-1],1,n,l,k,d);
      }
      if(str[0]=='Q')
      {
         scanf("%d%d",&l,&k);
         long long v=query(root[t],1,n,l,k,0);
         printf("%lld\n",v);
      }
      if(str[0]=='H')
      {
         scanf("%d%d%d",&l,&k,&d);
         long long v=query(root[d],1,n,l,k,0);
         printf("%lld\n",v);
      }
      if(str[0]=='B')
      {
         scanf("%d",&l);
         t=l;
      }
   }
}
```

28. Persistent DSU (not validated)

```
#include <bits/stdc++.h>

#define DB(a) cerr << __LINE__ << ": " << #a << " = " << (a) << endl;

using namespace std;

/*  <persistent array>  */
const int MAXN = 1e5 + 5, MAXQ = MAXN, MAXS = 1e8;
int lch[MAXS], rch[MAXS], cnt;

int new_node (int l, int r)
{
   assert(cnt < MAXS);
   lch[cnt] = l;
   rch[cnt] = r;
   return cnt++;
}

struct p_array
{
   int n, root;

   int build (int *a, int n)
   {
      if (n == 1)
         return new_node(*a, *a);

      int m = n / 2;
      return new_node(build(a, m), build(a + m, n - m));
   }

p_array (int *a, int n) : n(n)
{
   root = build(a, n);
}

p_array (int n = 0, int root = 0) : n(n), root(root)
{}

int get (int v, int n, int i)
{
   if (n == 1)
      return lch[v];

   int m = n / 2;

   return i < m ? get(lch[v], m, i) : get(rch[v], n - m, i - m);
}

// get the value at potition i.
int operator [] (int i)
{
   return get(root, n, i);
}

int set (int v, int n, int i, int x)
{
   if (i < 0 || i >= n)
      return v;
```

```
    if (n == 1)
        return new_node(x, x);

    int m = n / 2;
    return new_node(set(lch[v], m, i, x), set(rch[v], n - m, i - m, x));
  }

  // get the resultant array of setting value x to position i.
  p_array set (int i, int x)
  {
    return p_array(n, set(root, n, i, x));
  }
}root[MAXQ];   // root[v] = root of the tree that represent the version # v
of the array.

/*  </persistent array>   */

int n, k, a, b, v, data[MAXN];
char c;

int find_set (p_array &data, int a)
{
  int p = data[a];

  if (p < 0)
    return a;

  int ans = find_set(data, p);
  //data = data.set(a, ans);
  return ans;
}

p_array union_set (p_array &data, int a, int b)
{
  a = find_set(data, a);
  b = find_set(data, b);

  if (a != b)
  {
    int cnt_a = data[a], cnt_b = data[b];

    if (cnt_a > cnt_b)
      swap(a, b);

    p_array ans = data.set(a, cnt_a + cnt_b);
    ans = ans.set(b, a);
    return ans;
  }
  else
    return data;
}

int main ()
{
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  cin >> n >> k;
  memset(data, -1, sizeof data);
  root[0] = p_array(data, n + 1);

  for (int i = 1; i <= k; ++i)
```

```
  {
    cin >> c >> v >> a >> b;

    if (c == '+')
      root[i] = union_set(root[v], a, b);
    else
    {
      int id_b = find_set(root[v], b);
      int id_a = find_set(root[v], a);
      cout << ((id_a == id_b) ? "YES" : "NO") << '\n';
    }
  }

  return 0;
}
```

29. Segment Tree (2D)

```
#include<bits/stdc++.h>
using namespace std;
#define D(x)    cout << #x " = " << (x) << endl
#define MAX     1005
#define xx      first
#define yy      second
typedef pair<int,int> pii;

const int inf = 1000000000;
struct segTree {
  int arr[MAX << 2];

  segTree() {
    for(int i = 0; i < (MAX << 2); i++) arr[i] = -inf;
  }

  void update(int idx, int st, int ed, int pos, int val,
                                        vector<int> &nodeList) {
    nodeList.push_back(idx);

    if(st == ed) {
      arr[idx] = max(arr[idx], val);
      return;
    }

    int mid = (st + ed)/2, l = idx << 1, r = l | 1;
    if(pos <= mid) update(l, st, mid, pos, val, nodeList);
    else update(r, mid+1, ed, pos, val, nodeList);

    arr[idx] = max(arr[l], arr[r]);
  }

  int query(int idx, int st, int ed, int i, int j) {
    if(st == i && ed == j) return arr[idx];

    int mid = (st + ed)/2, l = idx << 1, r = l | 1;
    if(j <= mid) return query(l, st, mid, i, j);
    if(i > mid) return query(r, mid+1, ed, i, j);
    else return max(query(l, st, mid, i, mid),
                    query(r, mid+1, ed, mid+1, j));
  }
};

struct _2DsegTree {
  segTree segArr[MAX << 2];
```

```
  vector<int> affected_nodes;

  void update(int idx, int st, int ed, int i, int j, int val) {
    if(st == ed) {
      affected_nodes.clear();
      segArr[idx].update(1, 1, MAX, j, val, affected_nodes);
      return;
    }

    int mid = (st + ed)/2, l = idx << 1, r = l | 1;
    if(i <= mid) update(l, st, mid, i, j, val);
    else update(r, mid+1, ed, i, j, val);

    for(int p = 0; p < (int) affected_nodes.size(); p++) {
      int q = affected_nodes[p];
      segArr[idx].arr[q] = max(segArr[l].arr[q], segArr[r].arr[q]);
    }
  }


  int query(int idx, int st, int ed, int st_r, int ed_r, int st_c, int ed_c) {
    if(st == st_r && ed == ed_r)
        return segArr[idx].query(1, 1, MAX, st_c, ed_c);

    int mid =(st + ed)/2, l =  idx << 1, r = l + 1;
    if(ed_r <= mid) return query(l, st, mid, st_r, ed_r, st_c, ed_c);
    if(st_r > mid) return query(r, mid+1, ed, st_r, ed_r, st_c, ed_c);
    return max(query(l, st, mid, st_r, mid, st_c, ed_c),
                    query(r, mid+1, ed, mid+1, ed_r, st_c, ed_c));
  }
};
```

```
_2DsegTree tree;
vector<pii> input;

int main() {
  int i, x, y, n, mx = 1;

  scanf("%d", &n);
  for(i = 1; i <= n; i++) {
    scanf("%d %d", &x, &y);
    input.push_back(pii(x,y));
  }

  for(i = n - 1; i >= 0; i--) {
    x = input[i].xx;
    y = input[i].yy;
    int q = 1 + max(0, tree.query(1, 1, MAX, x, MAX, y, MAX));
    tree.update(1, 1, MAX, x, y, q);

    mx = max(mx, q);
  }

  printf("%d\n", mx);
  return 0;
}
```

30. 2-D Range Update

```
#define ll long long

char str[500];
```

```
ll tree[4][1005][1005];
int n;
void update(int x,int y,int val,int i)
{
   int y1;
   while(x<=n)
   {
      y1=y;
      while(y1<=n)
      {
         tree[i][x][y1]+=val;
         y1+=(y1&-y1);
      }
      x+=(x&-x);
   }
}
long long query(int x,int y,int i)
{
   long long ans=0;
   int y1;
   while(x>0)
   {
      y1=y;
      while(y1>0)
      {
         ans+=tree[i][x][y1];
         y1-=(y1&-y1);
      }
      x-=(x&-x);
   }
   return ans;
}
void updater(int x1,int y1,int x2,int y2,int k)
{
   update(x1,y1,k,0);
   update(x1,y2+1,-k,0);
   update(x2+1,y1,-k,0);
   update(x2+1,y2+1,k,0);

   update(x1,y1,(1-y1),1);
   update(x1,y2+1,y2,1);
   update(x2+1,y1,(y1-1),1);
   update(x2+1,y2+1,-y2,1);

   update(x1,y1,k*(1-x1),2);
   update(x1,y2+1,k*(x1-1),2);
   update(x2+1,y1,x2,2);
   update(x2+1,y2+1,-x2,2);

   update(x1,y1,(x1-1)*(y1-1),3);
   update(x1,y2+1,-y2*(x1-1),3);
   update(x2+1,y1,-x2*(y1-1),3);
   update(x2+1,y2+1,x2*y2,3);
}
long long queryman(int x1,int y1,int x2,int y2)
{
   ll
val1=query(x2,y2,0)*x2*y2+query(x2,y2,1)*x2+query(x2,y2,2)*y2+query(
x2,y2,3);
   ll val2=query(x2,y1-1,0)*x2*(y1-1)+query(x2,y1-1,1)*x2+query(x2,y1-
1,2)*(y1-1)+query(x2,y1-1,3);
```

```
   ll val3=query(x1-1,y2,0)*(x1-1)*y2+query(x1-1,y2,1)*(x1-1)+query(x1-
1,y2,2)*y2+query(x1-1,y2,3);
   ll val4=query(x1-1,y1-1,0)*(x1-1)*(y1-1)+query(x1-1,y1-1,1)*(x1-
1)+query(x1-1,y1-1,2)*(y1-1)+query(x1-1,y1-1,3);

   return val1-val2-val3+val4;
}
int main()
{
   int i,j,k,m,x1,y1,x2,y2;
   scanf("%d",&k);
   for(j=1; j<=k; j++)
   {
      memset(tree,0,sizeof tree);
      scanf("%d%d",&n,&m);
      while(m--)
      {
         scanf("%s",&str);
         if(str[0]=='C')
         {
            scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
            updater(x1,y1,x2,y2,1);
            continue;
         }
         scanf("%d%d",&x1,&y1);
         int ans=queryman(x1,y1,x1,y1);
         printf("%d\n",ans%2);
      }
      if(j!=k)
         printf("\n");
   }
}
```

31. Sparse Table

```
#define Max 10000005
int ST[24][Max];
int A[Max];
void Compute_ST(int N) {
   for (int i=0; i<N; i++)ST[0][i] = i;
   for (int k = 1; (1 << k)<N; k++) {
      for (int i=0; i+(1<<k)<=N; i++) {
         int x = ST[k-1][i];
         int y = ST[k-1][i+(1<<k-1)];
         ST[k][i]=A[x]<=A[y]?x:y;
      }
   }
}
int RMQ(int i, int j) {
   int k = log2(j-i);
   int x = ST[k][i];
   int y = ST[k][j-(1<<k)+1];
   return A[x] <= A[y] ? x : y;
}
int main() {
   int N;
   cin>>N;
   for(int i=0; i<N; i++) {
      cin>>A[i];
   }
   Compute_ST(N);
   int Q;
   cin>>Q;
```

```
    while(Q--) {
        int x,y;
        cin>>x>>y;
        cout<<A[RMQ(x,y)]<<endl;
    }
    return 0;
}
```

32. RMQ

```
int ara[505][505],much[505][505][12];
int main()
{
    int i,j,k,l,m,n,q,test,casio=1;
    scanf("%d",&test);
    while(test--)
    {
        scanf("%d%d",&n,&q);
        for(i=1; i<=n; i++)
        {
            for(j=1; j<=n; j++)
            {
                scanf("%d",&ara[i][j]);
                much[i][j][0]=ara[i][j];
            }
        }
        for(i=1; i<=n; i++)
        {
            for(int j=1; (1<<j)<=n; j++)
            {
                for(k=1; k+(1<<(j-1))<=n; k++)
                {
                    much[i][k][j]=max(much[i][k][j-1],much[i][k+(1<<(j-1))][j-1]);
                }

            }
        }
        int x,y,endx,endy,s;
        printf("Case %d:\n",casio++);
        while(q--)
        {
            scanf("%d%d%d",&x,&y,&s);
            endx=x+s-1;
            endy=y+s-1;
            k=log2(endy-y+1);
            int ma=0;
            for(int i=x; i<=endx; i++)
            {
                ma=max(ma,much[i][y][k]);
                ma=max(ma,much[i][endy-(1<<k)+1][k]);
            }
            printf("%d\n",ma);
        }
        memset(much,0,sizeof much);
    }
}
```

33. Treap (Shahriar)

```
struct item {
    int key, prior;
    int val, sum, lazy;
    int mx; /// mx value in this Treap Tree
    int repl;
```

```
    bool repl_flag;
    bool rev;
    item *l, *r;

    item() {}
    item(int _key, int _prior) {
        key = _key, prior = _prior;
        val = sum = 0;
        l = NULL,   r = NULL;
    }
};
typedef item* Treap;

/**
*   It'll return a new Treap node having value val
*/
Treap init(int val) {
    Treap node =(Treap)malloc(sizeof(item));
    node->key = 1;
    node->val = node->sum = val;
    node->mx = val;
    node->lazy = 0;
    node->repl = 0;
    node->repl_flag = false;
    node->rev = false;
    node->prior = rand();
    node->l = node->r = NULL;
    return node;
}
/**
*   It'll return the total size of current Treap node
*/
int cnt(Treap t) {
    if( t ) return t->key;
    return 0;
}
void upd_cnt(Treap &t) {
    if( t )
        t->key = cnt(t->l) + cnt(t->r) + 1;
}
void upd_lazy(Treap t) {
    if( !t or !t->lazy )    return;
    t->val += t->lazy;
    t->mx += t->lazy;
    t->sum = t->val*cnt(t);
    if( t->l )  t->l->lazy += t->lazy;
    if( t->r )  t->r->lazy += t->lazy;
    t->lazy = 0;
}
void upd_repl(Treap t) {
    if( !t or !t->repl_flag )   return;
    t->val = t->mx = t->repl;
    t->sum = t->val*cnt(t);
    if( t->l ) {
        t->l->repl = t->repl;
        t->l->repl_flag = true;
    }
    if( t->r ) {
        t->r->repl = t->repl;
        t->r->repl_flag = true;
    }
    t->repl_flag = false;
```

```
    t->repl = 0;
}
void upd_rev(Treap t) {
    if( !t or !t->rev ) return;
    t->rev = false;
    swap(t->l, t->r);
    if( t->l )  t->l->rev ^= true;
    if( t->r )  t->r->rev ^= true;
}
void reset(Treap t) {
    if( !t ) return;
    t->mx = t->val;
    t->sum = t->val;
}
void combine(Treap &t, Treap l, Treap r) {
    if( !l )    t = r;
    else if( !r )   t = l;
    else {
        t->mx = max(l->mx, r->mx);
        t->sum = l->sum + r->sum;
    }
}
void operation(Treap t) {
    if( !t )    return;
    reset(t);
    upd_rev(t->l);
    upd_rev(t->r);
    upd_repl(t->l);
    upd_repl(t->r);
    upd_lazy(t->l);
    upd_lazy(t->r);
    combine(t, t->l, t);
    combine(t, t, t->r);
}
void split(Treap t, Treap &l, Treap &r, int key, int add = 0) {
    upd_rev(t);
    upd_repl(t);
    upd_lazy(t);
    if( !t )
        return void (l = r = 0);
    int cur_key = add + cnt(t->l);
    if( key<=cur_key ) {
        split(t->l, l, t->l, key, add);
        r = t;
    } else {
        split(t->r, t->r, r, key, add+1+cnt(t->l));
        l = t;
    }
    upd_cnt(t);
    operation(t);
}
void Merge(Treap &t, Treap l, Treap r) {
    upd_rev(l);
    upd_rev(r);
    upd_repl(l);
    upd_repl(r);
    upd_lazy(l);
    upd_lazy(r);
    if( !l ) {
        t = r;
        return;
    }
```

```
    if( !r ) {
        t = l;
        return;
    }
    if( l->prior > r->prior ) {
        Merge(l->r, l->r, r);
        t = l;
    } else {
        Merge(r->l, l, r->l);
        t = r;
    }
    upd_cnt(t);
    operation(t);
}
/**
*   It'll add v to all the elements from l to r
*   1-indexing
*/
void range_update(Treap t, int l, int r, int v) {
    Treap lft, mid;
    split(t, lft, t, l-1);
    split(t, mid, t, r-l+1);
    upd_repl(mid);
    mid->lazy = v;
    Merge(t, mid, t);
    Merge(t, lft, t);
}
/**
*   It'll replace all the elements to v from l to r
*   1-indexing
*/
void range_replace(Treap t, int l, int r, int v) {
    Treap lft, mid;
    split(t, lft, t, l-1);
    split(t, mid, t, r-l+1);
    upd_lazy(mid);
    mid->repl_flag = true;
    mid->repl = v;
    Merge(t, mid, t);
    Merge(t, lft, t);
}
/**
*   It'll Reverse the elements from l to r
*   1-indexing
*/
void range_reverse(Treap t, int l, int r) {
    Treap lft, mid;
    split(t, lft, t, l-1);
    split(t, mid, t, r-l+1);
    mid->rev ^= true;
    Merge(t, mid, t);
    Merge(t, lft, t);
}
/**
*   It'll return the sum of all elements from l to r
*   1-indexing
*/
int range_query(Treap t, int l, int r) {
    Treap lft, mid;
    split(t, lft, t, l-1);
    split(t, mid, t, r-l+1);
    int ret = mid->sum;
```

```
    Merge(t, mid, t);
    Merge(t, lft, t);
    return ret;
}
Treap Root; /// Root of Treap
int main() {
    int v, n;
    scanf("%d", &n);
    for( int i = 0; i<n; i++ ) {
        /// All the elements will be inserted to Treap one by one
        /// automatically 1-indexing
        scanf("%d", &v);
        if( !i )    Root = init(v);
        else    Merge(Root, Root, init(v));
    }
    int q, typ, p, l, r;
    scanf("%d", &q);
    while(q--) {
        scanf("%d", &typ);
        if( typ == 0 ) { /// sum of all elements from l to r
            scanf("%d %d", &l, &r);
            /// range_query() can be modified to get other data of this Range
            int ans = range_query(Root, l, r);
            printf("%d\n", ans);
        } else if( typ == 1 ) { /// Add v to all elements from l to r
            scanf("%d %d %d", &l, &r, &v);
            range_update(Root, l, r, v);
        } else if( typ == 2 ) { /// Replace all elements from l to r by v
            scanf("%d %d %d", &l, &r, &v);
            range_replace(Root, l, r, v);
        } else if( typ == 3 ) { /// Reverse all elements from l to r
            scanf("%d %d", &l, &r);
            range_reverse(Root, l, r);
        } else if( typ == 4 ) { /// Replace p-th element by v
            scanf("%d %d", &p, &v);
            range_replace(Root, p, p, v);
        }
    }
    return 0;
}
```

34. Treap (Jubair)

```
int del,big=-2000000000;
typedef struct node
{
    int prior,size;
    int val;//value stored in the array
    int sum,lsum,rsum,maxsum;//whatever info you want to maintain in
segtree for each node
    struct node *l,*r;
} node;
typedef node* pnode;
int cals(pnode t)
{
    return t?t->sum:0;
}
int call(pnode t)
{
    return t?t->lsum:0;
}
int calr(pnode t)
{
    return t?t->rsum:0;
```

```
}
int calm(pnode t)
{
   return t?t->maxsum:big;
}
int sz(pnode t)
{
   return t?t->size:0;
}
void upd_sz(pnode t)
{
   if(t)t->size=sz(t->l)+1+sz(t->r);
}
void reset(pnode t)
{
   if(t)t->sum=t->lsum=t->rsum=t->maxsum=t->val;//no need to reset lazy
coz when we call this lazy would itself be propagated
}
void combine(pnode &t) //combining two ranges of segtree
{
  if(!t)return ;
   t->sum=cals(t->l)+t->val+cals(t->r);

   t->lsum=max(max(call(t->l),cals(t->l)+t->val),cals(t->l)+t->val+call(t->r));

   t->rsum=max(max(calr(t->r),cals(t->r)+t->val),cals(t->r)+t->val+calr(t-
>l));

   t->maxsum=max(max(calm(t->l),calm(t->r)),calr(t->l)+t->val+call(t->r));
   t->maxsum=max(t->maxsum,t->val);
}
void operation(pnode t)
{
   if(!t)return;
   reset(t);
   combine(t);
}
void split(pnode t,pnode &l,pnode &r,int pos,int add=0)
{
   if(!t)return void(l=r=NULL);
   int curr_pos = add + sz(t->l);
   if(curr_pos<=pos){
      split(t->r,t->r,r,pos,curr_pos+1);
      l=t;
   }
   else{
      split(t->l,l,t->l,pos,add);
      r=t;
   }
   upd_sz(t);
   operation(t);
}
void merge(pnode &t,pnode l,pnode r)  //l->leftarray,r->rightarray,t-
>resulting array
{
   if(!l || !r) t = l?l:r;
   else if(l->prior>r->prior)merge(l->r,l->r,r),t=l;
   else    merge(r->l,l,r->l),t=r;
   upd_sz(t);
   operation(t);
}
pnode init(int val)
{
   pnode ret = (pnode)malloc(sizeof(node));
   ret->prior=rand();
   ret->size=1;
```

```
    ret->val=val;
    ret->sum=ret->lsum=ret->rsum=ret->maxsum=val;
    ret->l=ret->r=NULL;
    return ret;
}
int range_query(pnode t,int l,int r) //[l,r]
{
    pnode L,mid,R;
    split(t,L,mid,l-1);
    split(mid,t,R,r-l);//note: r-l!!
    int ans = t->maxsum;
    merge(mid,L,t);
    merge(t,mid,R);
    return ans;
}
void insert(pnode &t,int pos,int val)
{
    pnode L,R,mid,temp;
    pnode it=init(val);
    split(t,mid,R,pos-1);
    merge(temp,mid,it);
    merge(t,temp,R);
}
void erase(pnode &t,int pos)
{
    pnode L,R,mid,temp;
    split(t,mid,R,pos);
    split(mid,L,temp,pos-1);
    merge(t,L,R);
   if(temp) delete temp;
}
int main()
{
    int i,j,k,l,m,n,q;
    pnode root=NULL;
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        del=i;
        scanf("%d",&l);
        insert(root,i,l);
       // cout<<root->maxsum<<endl;
        //  cout<<root->val<<endl;
    }
    char str[5];
    scanf("%d",&q);
    while(q--)
    {
        scanf("%s",str);
        if(str[0]=='I')
        {
            scanf("%d%d",&l,&k);
            l--;
            insert(root,l,k);
            continue;
        }
        if(str[0]=='D')
        {
            scanf("%d",&l);
            l--;
      //     del=49;
            erase(root,l);
            continue;
        }
        if(str[0]=='R')
        {
```

```
        scanf("%d%d",&l,&k);
        l--;
        erase(root,l);
        insert(root,l,k);
        continue;
      }
      scanf("%d%d",&l,&k);
      l--,k--;
      l=range_query(root,l,k);
      printf("%d\n",l);
   }
}
```

35. LCA 1

```
#define MXN 100000
#define SZE 17
int n, q, dp[MXN+3][SZE+3], level[MXN+7];
vector<int> adj[MXN+7];
void dfs(int u, int pre) {
   for( int i = 1; i<=SZE; i++ ) {
      dp[u][i] = dp[dp[u][i-1]][i-1];
   }
   for( auto v:adj[u] ) {
      if( v == pre )  continue;
      dp[v][0] = u;
      level[v] = level[u]+1;
      dfs(v, u);
   }
}
int lca(int u, int v) {
   if( level[u]>level[v] ) swap(u, v);
   for( int i = SZE; i>=0; i-- ) {
      int par = dp[v][i];
      if( level[par]>=level[u] ) {
         v = par;
      }
   }
   if( u == v )    return u;
   for( int i = SZE; i>=0; i-- ) {
      if( dp[u][i] != dp[v][i] ) {
         u = dp[u][i];
         v = dp[v][i];
      }
   }
   return dp[u][0];
}

int main() {
//   freopen("H:\\00.txt", "r", stdin);
   int u, v, i, j, k;
   sf2(n, q);
   for( i = 1; i<n; i++ ) {
      sf2(u, v);
      adj[u].pb(v);
      adj[v].pb(u);
   }
   level[1] = 0;
   dp[1][0] = 1;
   dfs(1, 0);
   for( i = 0; i<q; i++ ) {
      sf2(u, v);
      printf("%d %d : %d\n", u, v, lca(u, v));
   }
```

```
    return 0;
}
```

36. LCA 2

```
///complexity build(V)
///per query complexity(strictly log(step))
///graph must be tree
int node, T;
int parent[MX][step+1];
int start[MX], finish[MX];
int distan[MX], level[MX];
vector<data> adj[MX];

void dfs(int u, int p, int dis, int lev) {
    start[u] = T++;
    level[u] = lev;
    distan[u] = dis;
    parent[u][0] = p;
    for(int i = 1; i <= step; i++)
        parent[u][i] = parent[parent[u][i-1]][i-1];
    for(int i = 0; i < adj[u].size(); i++) {
        v = adj[u][i].v;
        if(v != p)
            dfs(v,u,dis+adj[u][i].w,lev+1);
    }
    finish[u] = T++;
    return;
}
bool Is_Anchestor(int u, int v) {
    if(start[u] <= start[v] && finish[u] >= finish[v])
        return true;
    return false;
}
int LCA_Query(int u, int v) {
    if(Is_Anchestor(u,v)) return u;
    if(Is_Anchestor(v,u)) return v;
    int tem = u;
    for(int i = step; i>=0; i--)
        if(!Is_Anchestor(parent[tem][i],v))
            tem = parent[tem][i];
    return parent[tem][0];
}
int Kth_Query(int u, int k) {
    int tem = u;
    for(int i = step; i>=0; i--)
        if((k>>i)&1 == 1)
            tem = parent[tem][i];
    return tem;
}
void lca_cls(void) {
    for(int i = 0; i <= node; i++) {
        adj[i].clear();
        for(int j = 0; j <= step; j++)
            parent[i][j] = 1;
    }
}
void input() {
    scanf("%d", &node);
    lca_cls();
    for(int i = 1; i < node; i++) {
        scanf("%d%d%d", &u, &v, &w);
        adj[u].push_back(data(v,w));
```

```
        adj[v].push_back(data(u,w));
    }
}
int main() {
    input();
    T = 0;
    dfs(1,1,0,1);
    lca = LCA_Query(u,v);
    res = distan[u]+distan[v]-2*distan[lca];
    ///distance
    w--;///find w'th node
    if(level[u]-level[lca] == w) res = lca;
    else if(level[u]-level[lca] > w)
        res = Kth_Query(u,w);
    else
        res = Kth_Query(v,level[u]+level[v]-2*level[lca]-w);
}
```

37. HLD

```
int indx, tot_chain, chain_no[MXN+7]; /// for HLD
int chain_head[MXN+7], pos_in_base[MXN+7];
int base_arr[MXN+7];
int depth[MXN+7], cost[MXN+7];/// for Tree
int son[MXN+7], prnt[MXN+7];
int n, tree[4*MXN+7]; /// for segment tree

vector<int> adj[MXN+7]; /// for Graph

void reset(int num) {
    indx = -1;
    tot_chain = 0;
    for( int i = 0; i<=num; i++ ) {
        adj[i].clear();
        chain_head[i] = son[i] = -1;
    }
}
/// Start of Segment Tree
int build(int node, int base, int top) {
    if( base==top ) {
        tree[node] = cost[base_arr[base]];
        return tree[node];
    }
    int left, right, mid;
    left = node<<1;
    right = left+1;
    mid = ( base+top )>>1;
    tree[node] = build(left, base, mid);
    tree[node] += build(right, mid+1, top);
    return tree[node];
}
int update(int node, int base, int top, int i, int v) {
    if( base>i or top<i )   return tree[node];
    if( base==top ) {
        tree[node] = v;
        return tree[node];
    }
    int left, right, mid;
    left = node<<1;
    right = left+1;
    mid = ( base+top )>>1;
    tree[node] = update(left, base, mid, i, v);
    tree[node] += update(right, mid+1, top, i, v);
```

```
    return tree[node];
}
int query(int node, int base, int top, int i, int j) {
    if( base>j or top<i )   return 0;
    if( base>=i and top<=j )    return tree[node];
    int left, right, mid;
    left = node<<1;
    right = left+1;
    mid = ( base+top )>>1;
    int ret = query(left, base, mid, i, j);
    ret += query(right, mid+1, top, i, j);
    return ret;
}
/// End of Segment Tree

/**
*   dfs() used to set parent of a node,
*   depth of a node, special son of a node
*   it will return the subtree size of node - u
*/
int dfs(int u, int pre) {
    prnt[u] = pre;
    int mx = -1, ret = 1;
    for( int i = 0; i<adj[u].size(); i++ ) {
        int v = adj[u][i];
        if( v == pre )  continue;
        depth[v] = depth[u] + 1;
        int sz = dfs(v, u);
        ret += sz;
        if( sz>mx ) {
        /// For finding special son of node - u
            mx = sz;
            son[u] = v;
        }
    }
    return ret;
}

/**
*   Actual HLD Part
*/
void HLD(int u, int pre) {
    if( chain_head[tot_chain] == -1 )
        chain_head[tot_chain] = u;
    pos_in_base[u] = ++indx;
    base_arr[indx] = u;
    chain_no[u] = tot_chain;
    if( son[u] == -1 )  return;
    HLD(son[u], u);
    for( int i = 0; i<adj[u].size(); i++ ) {
        int v = adj[u][i];
        if( v == pre or v == son[u] )   continue;
        tot_chain++;
        HLD(v, u);
    }
}

/**
*   It'll return sum of cost
*   from node - u to node - v
*/
int solve(int u, int v) {
```

```
    int ch1 = chain_no[u];
    int ch2  = chain_no[v];
        /// chd_u = Chain Head of u_chain
    int chd_u = chain_head[ch1];
        /// chd_v = Chain Head of v_chain
    int chd_v = chain_head[ch2];
    int ret = 0;
        /// while two chains are in different chain
    while( chd_u != chd_v ) {
      if( depth[chd_u]<depth[chd_v] ) {
            /// So that u-head will always deeper than v-head
        swap(chd_u, chd_v);
        swap(u, v);
      }
      ret += query(1, 0, indx, pos_in_base[chd_u], pos_in_base[u]);
      u = prnt[chd_u];    /// Changing the Chain
      ch1 = chain_no[u];
      chd_u = chain_head[ch1];  /// updating chain head
    }
    if( depth[u]<depth[v] ) swap(u, v);
    ret += query(1, 0, indx, pos_in_base[v], pos_in_base[u]);
    return ret;
}

int main() {
    //freopen("H:\\00.txt", "r", stdin);
    int t, cas = 0, q, typ, u, v, pos, ans, i, j, k;
    scanf("%d", &t);
    while(t--) {
      scanf("%d", &n);
      reset(n);
      for( i = 0; i<n; i++ ) {
        scanf("%d", &cost[i]); /// i-th node, indexing 0
      }
      for( i = 1; i<n; i++ ) {
        scanf("%d %d", &u, &v);
        adj[u].push_back(v);
        adj[v].push_back(u);
      }
      depth[0] = 0; /// 0- is Root, depth = 0
      dfs(0, -1);
      HLD(0, -1);
      build(1, 0, indx);
      printf("Case %d:\n", ++cas);
      scanf("%d", &q);
      while(q--) {
        scanf("%d", &typ);
        if( typ == 1 ) {
          scanf("%d %d", &u, &v);
          cost[u] = v;
          update(1, 0, indx, pos_in_base[u], v);
        } else {
          scanf("%d %d", &u, &v);
          ans = solve(u, v);
          printf("%d\n", ans);
        }
      }
    }
    return 0;
}
```

38. Diametre of a Tree in O(NlogN)

```
#include<bits/stdc++.h>
using namespace std;
const int mx=1e5+10,mxl=20;

int ar[mx],vis[mx];
vector<int>g[mx];
vector<long long>w[mx];
int rmq[mxl][mx+mx];
int fst[mx],lst[mx];
int lyr[mx];
long long dep[mx];
vector<int>eul;

void dfs(int u,int pu)
{
   eul.push_back(u);
   for(int i=0; i<int(g[u].size()); i++)
   {
      int v=g[u][i];
      long long c=w[u][i];
      if(v!=pu)
      {
         lyr[v]=1+lyr[u];
         dep[v]=dep[u]+c;
         dfs(v,u);
         eul.push_back(u);
      }
   }
}
int minimo(int l,int r)
{
   int len=32-__builtin_clz(r-l+1)-1;
   int x=rmq[len][l];
   int y=rmq[len][r-(1<<len)+1];
   if(lyr[x]<lyr[y])return x;
   return y;
}
int lca(int u,int v)
{
   if(lst[u]>lst[v])swap(u,v);
   int from=lst[u];
   int to=fst[v];
   if(from>to)to=lst[v];
   return minimo(from,to);
}

struct diameter
{
   int t[2];
   long long l;
   bool operator <(diameter d)const
   {
      return l<d.l;
   }
} diam[mx];
diameter join(diameter a,diameter b)
{
   diameter ans=max(a,b);
   for(int i=0; i<2; i++)
   {
```

```
      for(int j=0; j<2; j++)
      {
         int u=a.t[i];
         int v=b.t[j];
         int z=lca(u,v);
         long long len=dep[u]+dep[v]-2ll*dep[z];
         ans=max(ans, {u,v,len});
      }
   }
   return ans;
}

int p[mx];
int gp(int u)
{
   return (u==p[u]?u:p[u]=gp(p[u]));
}

const int mxa=1e4+10;
vector<int>divs[mxa];
vector< pair<int,int> >nodes[mxa];
int main()
{
   for(int i=1; i<mxa; i++)
   {
      for(int j=i; j<mxa; j+=i)
         divs[j].push_back(i);
   }

   int t,n;
   scanf("%d",&t);
   while(t--)
   {
      scanf("%d",&n);
      for(int i=0; i<n; i++)
      {
         scanf("%d",&ar[i]);
         g[i].clear();
         w[i].clear();
         for(int d : divs[ar[i]])
            nodes[d].clear();
         fst[i]=lst[i]=-1;
      }
      for(int i=0; i<n; i++)
      {
         for(int d : divs[ar[i]])
            nodes[d].push_back({ar[i],i});
      }
      for(int i=0; i<n-1; i++)
      {
         int u,v;
         long long c;
         scanf("%d %d %lld",&u,&v,&c);
         u--,v--;
         g[u].push_back(v);
         g[v].push_back(u);
         w[u].push_back(c);
         w[v].push_back(c);
      }

      eul.clear();
      dfs(0,0);
```

```
int es=eul.size();
for(int i=0; i<es; i++)
{
   int ei=eul[i];
   rmq[0][i]=ei;
   if(fst[ei]==-1)fst[ei]=i;
   lst[ei]=i;
}
for(int l=1; l<mxl; l++)
{
   for(int i=0; i<es; i++)
   {
      rmq[l][i]=rmq[l-1][i];
      int j=i+(1<<(l-1));
      if(j<es && lyr[rmq[l-1][j]]<lyr[rmq[l][i]])
         rmq[l][i]=rmq[l-1][j];
   }
}

long long ans=0;
for(int i=1; i<=10000; i++)
{
   if((int)nodes[i].size()==0) continue;

   sort(nodes[i].rbegin(),nodes[i].rend());
   for(int j=0; j<nodes[i].size(); j++)
   {
      int u=nodes[i][j].second;
      p[u]=u;
      vis[u]=1;
      diam[u]= {{u,u},0};//dimetre initialize korlam
      }

      for(int j=0; j<nodes[i].size(); j++)
      {
         int u=nodes[i][j].second;
         int mn=nodes[i][j].first;
         for(int v : g[u])
         {
            if(vis[v]==0 && ar[v]%i==0)
            {
               //we are merge two trees having node u and v and putting
edge between u and v
               int paru=gp(u);
               int parv=gp(v);
               if(paru==parv) continue;
               p[parv]=paru;
               diameter d=join(diam[paru],diam[parv]);
               diam[paru]=d;
               ans=max(ans,1ll*mn*1ll*i*diam[paru].l);
            }
         }
         vis[u]=0;
      }
    }
    printf("%lld\n",ans);
  }
  return 0;
}
```

39. MO's on Tree

```
vector<int> graph[sz] ;
int arr[sz], depth[sz], timer, par[sz][20], strt[sz], close[sz] ;

void dfs(int u,int p,int dep) {
   depth[u] = dep ;
   par[u][0] = p ;
   arr[++timer] = u ;
   strt[u] = timer ;

   for(int v : graph[u]) {
      if(v == p)
         continue;
      dfs(v,u,dep+1);
   }

   arr[++timer] = u ;
   close[u] = timer;
}

void build_lca(int n) {
   // build LCA Table
}

int lca(int u,int v) {
   return LCA ;
}

struct data {
   int l, r, box, id, ca ;

   data() {}

   data(int l,int r,int box,int id,int ca) {
      this->l = l ;
      this->r = r ;
      this->box = box ;
      this->id = id ;
      this->ca = ca ;
   }
};
bool cmp(data a,data b) {
   if(a.box == b.box)
      return a.r<b.r ;

   return a.box < b.box ;
}

data q_list[sz] ;
long sum, ans[sz] ;
bool flag[sz] ;

void add_list(int u) {
   // if flag[u] == false you should add this node
   // otherwise remove this node

   flag[u] = !flag[u] ;
}

void remove_list(int u) {
   // if flag[u] == false you should add this node
```

```
   // otherwise remove this node
}
void solve(int n) {
   memset(par,-1,sizeof par);
   dfs(1,-1,0);
   build_lca(n);

   int q, box = sqrt(timer) + 3, p ;

   scanf("%d",&q);

   for(int i=0 ; i<q ; i++) {
      scanf("%d %d",&u,&v);

      if(strt[v]<strt[u])
         swap(u,v);

      p = lca(u,v);

      q_list[i].ca = p ;
      q_list[i].id = i ;

      if(p == u) {
         q_list[i].l = strt[u] ;
         q_list[i].r = strt[v] ;
         q_list[i].box = q_list[i].l/box ;
      }

      else {
         q_list[i].l = close[u] ;
         q_list[i].r = strt[v] ;
         q_list[i].box = q_list[i].l/box ;
      }
   }

   sort(q_list,q_list+q,cmp);

   int l = 1, r = 1 ;

   for(int i=0 ; i<q ; i++) {
      while(r<=q_list[i].r) {
         add_list(arr[r]);
         r++;
      }

      while(r>q_list[i].r+1) {
         remove_list(arr[r-1]);
         r--;
      }

      while(l>q_list[i].l) {
         add_list(arr[l-1]);
         l--;
      }

      while(l<q_list[i].l) {
         remove_list(arr[l]);
         l++;
      }

      ans[q_list[i].id] = sum  ; // here sum is ans
   }
```

```
}
```

40. MO's with Update

```
int dx[]= {1,-1,0,0,-1,-1,1,1};/*8 move*/
int dy[]= {0,0,1,-1,1,-1,1,-1};/*8 move*/
//int dx[]={1,1,2,2,-1,-1,-2,-2};/*knight move*/
//int dy[]={2,-2,1,-1,2,-2,1,-1};/*knight move*/
#define MXK        450000
#define mx 400000
#define jora       pair<ll , ll>
#define fs         first
#define sc         second
#define ll long long
#define mod 10007

ll sz[mx],cnt[mx],ans,ara[mx],ou[mx],rev[mx];
int currentL=0,currentR=-1;
struct edge
{
   int l,k,koyta,id,v,w;
   bool operator<(const edge &P)const
   {

      if(v==P.v)
      {
         if(w==P.w)
            return koyta<P.koyta;
         return w<P.w;
      }
      return v<P.v;
   }
};
struct node
{
   int age,pore,indx;
};
node my[MXK];
int get(double n)
{
   double a=2,b=3;
   a=a/b;
   return pow(n+.5,a);
}
void add(int pos)
{
   cnt[ara[pos]]++;
   if(cnt[ara[pos]]==1)
   {
      ans+=rev[ara[pos]];
   }
}
void remove(int pos)
{
   cnt[ara[pos]]--;
   if(cnt[ara[pos]]==0)
   {
      ans-=rev[ara[pos]];
   }
}
void addtime(int no)
{
   int age=my[no].age;
```

```
    int pore=my[no].pore;
    int pos=my[no].indx;
    if(pos>=currentL and pos<=currentR)
    {
        cnt[age]--;
        if(cnt[age]==0)
            ans-=rev[age];
    }
    if(pos>=currentL and pos<=currentR)
    {
        cnt[pore]++;
        if(cnt[pore]==1)
            ans+=rev[pore];
    }
    ara[my[no].indx]=pore;
}
void remtime(int no)
{
    int age=my[no].age;
    int pore=my[no].pore;
    int pos=my[no].indx;
    swap(age,pore);
    if(pos>=currentL and pos<=currentR)
    {
        cnt[age]--;
        if(cnt[age]==0)
            ans-=rev[age];
    }
    if(pos>=currentL and pos<=currentR)
    {
        cnt[pore]++;
        if(cnt[pore]==1)
            ans+=rev[pore];
    }
    ara[my[no].indx]=pore;
}
vector<int>ac;
vector<edge>vc;
map<int,int>name,pek;
int main()
{
    int i,j,k,l,m,n,nw=0,qes=0,cs=0,q;
    char ty[5];
    scanf("%d",&n);
    for(int i=0; i<n; i++)
    {
        scanf("%d",&ara[i]);
        ac.pb(ara[i]);
    }
    int Block=get(n);
    for(int i=0; i<n; i++)
    {
        sz[i]=(i+Block-1)/Block;
    }

    scanf("%d",&q);
    while(q--)
    {
        scanf("%s%d%d",&ty,&l,&k);
        l--;
        if(ty[0]=='U')
        {
```

```
        ac.pb(k);
        my[++nw].age=ara[l];
        my[nw].pore=k;
        my[nw].indx=l;
        ara[l]=k;
        continue;
    }
    k--;
    edge gr;
    gr.l=l;
    gr.k=k;
    gr.id=++qes;
    gr.koyta=nw;
    gr.v=sz[gr.l];
    gr.w=sz[gr.k];
    vc.pb(gr);
}
sort(ac.begin(),ac.end());
for(int i=0; i<ac.size(); i++)
{
    if(i&&ac[i]==ac[i-1])
        continue;
    pek[ac[i]]=++cs;
    rev[cs]=ac[i];
}
for(int i=1; i<=nw; i++)
{
    my[i].age=pek[my[i].age];
    my[i].pore=pek[my[i].pore];
}
for(int i=0; i<n; i++)
    ara[i]=pek[ara[i]];

sort(vc.begin(),vc.end());
for(int i=0; i<vc.size(); i++)
{
    int L=vc[i].l,R=vc[i].k,time=vc[i].koyta;
    while(nw<time)
    {
        addtime(++nw);
    }
    while(nw>time)
    {
        remtime(nw);
        nw--;
    }
    while(currentL<L)
    {
        remove(currentL);
        currentL++;
    }
    while(currentL>L)
    {
        add(currentL-1);
        currentL--;
    }
    while(currentR<R)
    {
        add(currentR+1);
        currentR++;
    }
    while(currentR>R)
```

```
    {
      remove(currentR);
      currentR--;
    }
    ou[vc[i].id]=ans;
  }
  for(int i=1; i<=qes; i++)
    printf("%lld\n",ou[i]);
}
```

41. Centroid Decomposition

```
#define sz 100000

struct Centroid_Decomposition {
  int node, lgn ;
  vector <vector <ll>> weight ;
  vector <vector <int>> par, graph ;
  vector <int> depth, subtree, cenp ;
  ll cost[sz], ans[sz], total_node ;
  const ll inf = 1e16 ;
  bool complete[sz] ;
  Centroid_Decomposition(int node):graph(node+7), weight(node+7),
depth(node+7), subtree(node+7), cenp(node+7) {
    this->node = node ;
    lgn = 1 + log2(node) ;
    par = vector <vector <int>>(node+7, vector <int>(lgn, -1));
    for(int i=0 ; i<=node ; i++)
      ans[i] = inf, complete[i] = false ;
  }
  void add_edge(int u,int v,ll w = 1) {
    graph[u].push_back(v);
    weight[u].push_back(w);
  }
  void dfs0(int u,int p,int dep,ll cst) {
    par[u][0] = p ;
    depth[u] = dep ;
    cost[u] = cst ;
    int v ;
    for(int i=0 ; i<graph[u].size() ; i++) {
      v = graph[u][i] ;
      if(v == p) continue;
      dfs0(v,u,dep+1,cst+weight[u][i]);
    }
  }
  void build_lca() {
    for(int i=1 ; i<lgn ; i++) {
      for(int j=1 ; j<=node ; j++) {
        if(par[j][i-1] != -1)
          par[j][i] = par[par[j][i-1]][i-1] ;
      }
    }
  }
  int lca(int u,int v) {
    if(depth[u] < depth[v])
      swap(u,v);
    for(int i=lgn-1 ; i>=0 ; i--) {
      if(par[u][i] != -1 && depth[par[u][i]] >= depth[v])
        u = par[u][i] ;
    }
    if(u == v)
      return u ;
    for(int i=lgn-1 ; i>=0 ; i--) {
```

```
        if(par[u][i] != par[v][i]) {
            u = par[u][i] ;
            v = par[v][i] ;
        }
    }
    return par[u][0] ;
}
ll dist(int u, int v) {
    int ca = lca(u,v);
    ll ans = cost[u] + cost[v] - (ll)2*cost[ca] ;
    return ans ;
}
void cal_subtree(int u,int p) {
    int v ;
    total_node++;
    subtree[u] = 1 ;
    for(int i=0 ; i<graph[u].size() ; i++) {
        v = graph[u][i] ;
        if(v != p && complete[v] == false) {
            cal_subtree(v,u);
            subtree[u] += subtree[v] ;
        }
    }
}
int centroid(int u,int p) {
    int v ;
    for(int i=0 ; i<graph[u].size() ; i++) {
        v = graph[u][i] ;
        if(v != p && complete[v] == false &&
                    subtree[v] > total_node/2LL)
            return centroid(v,u);
    }
    return u ;
}
void decomposition(int u,int p) {
    total_node = 0 ;
    cal_subtree(u,p);
    int cen = centroid(u,p);
    if(p == -1)
        p = cen ;
    cenp[cen] = p ;
    complete[cen] = true ;
    int v ;
    for(int i=0 ; i<graph[cen].size() ; i++) {
        v = graph[cen][i] ;
        if(v != p && complete[v] == false)
            decomposition(v,cen);
    }
}
void update(int u) {
    int x = u ;
    while(true) {
        ans[x] = min(ans[x],dist(u,x));
        if(x == cenp[x])
            return ;
        x = cenp[x] ;
    }
}
ll query(int u) {
    int x = u ;
    ll ret = 1000000000000000LL ;
    while(true) {
```

```
        ret = min(ret,ans[x]+dist(u,x));
        if(x == cenp[x])
            return ret ;
        x = cenp[x] ;
      }
   }
};
```

42. DSU on Tree

```
ll ans[sz], sum, mxm ;
bool big[sz] ;
int color[sz], cont[sz], sub[sz] ;
vector<int> graph[sz] ;
void cal_subtree(int u,int p) {
   int v ;
   sub[u] = 1 ;
   for(int i=0 ; i<graph[u].size() ; i++) {
      v = graph[u][i] ;
      if(v == p)
         continue ;
      cal_subtree(v,u);
      sub[u] += sub[v] ;
   }
}
void add(int u,int p,int x) {
   cont[color[u]] += x ;
   if(cont[color[u]] > mxm) {
      sum = color[u] ;
      mxm = cont[color[u]] ;
   }
   else if(cont[color[u]] == mxm)
      sum += color[u] ;
   int v ;
   for(int i=0 ; i<graph[u].size() ; i++) {
      v = graph[u][i] ;
      if(v == p or big[v])
         continue;
      add(v,u,x);
   }
}
void dsu(int u,int p,bool keep) {
   int v, mx = -1, bc = -1 ;
   for(int i=0 ; i<graph[u].size() ; i++) {
      v = graph[u][i] ;
      if(v == p)
         continue ;
      if(sub[v] > mx) {
         mx = sub[v] ;
         bc = v ;
      }
   }
   for(int i=0 ; i<graph[u].size() ; i++) {
      v = graph[u][i] ;
      if(v == p or v == bc)
         continue;
      dsu(v,u,0);
   }
   if(bc != -1) {
      dsu(bc,u,1);
      big[bc] = 1 ;
   }
   add(u,p,1);
```

```
    ans[u] = sum ;
    if(bc != -1)
        big[bc] = 0 ;
    if(keep == false) {
        add(u,p,-1);
        mxm = 0 ;
        sum = 0 ;
    }
}
```

43. Dynamic Connectivity + DSU with Remove

```
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;

#define MXN 600005
#define MXE
#define MXQ
#define SZE
#define MOD
#define EPS
#define INF 1000000000
#define MX  200
#define inf 100000000
#define mx

vector<pii>vc[1500000];
stack<int>st;
int siz[500000];
int ara[500000],dig[500000];
int par[500000],comp;

void update(int node,int st,int en,int l,int r,pii ps)
{
    if(st>r or en<l)
        return ;
    if(st>=l and en<=r)
    {
        vc[node].pb(ps);
        return ;
    }
    int mid=(st+en)>>1;
    update(node*2,st,mid,l,r,ps);
    update(node*2+1,mid+1,en,l,r,ps);
}
int findo(int u)
{
    while(par[u] != u)
        u = par[u];
    return u;

}
void unstack(int now)
{
    while(st.size()>now)
    {
        int u=st.top();
        st.pop();
        siz[par[u]]-=siz[u];
        par[u]=u;
        comp++;
    }
}
```

```
void unite(int u,int v)
{
   u= findo(u);
   v= findo(v);
   if(v==u)
      return ;
   if(siz[v]<siz[u])
   {
      swap(u,v);
   }
   comp--;
   siz[v]+=siz[u];
   par[u]=v;
   st.push(u);
}
void query(int node,int s,int e)
{
   if(s>e)
      return ;
   int sz=st.size();
   for(int i=0; i<vc[node].size(); i++)
   {
      pii gs=vc[node][i];
      unite(gs.first,gs.second);
   }
   if(s==e)
   {
      ara[s]=comp;
      unstack(sz);
      return ;
   }
   int mid=(s+e)>>1;
   query(node*2,s,mid);
   query(node*2+1,mid+1,e);
   unstack(sz);
   return ;
}
set<pii>s;
set<pii>::iterator it;
map<pii,int>strt;
char str[30];
int main()
{
   freopen("connect.in", "r", stdin);
   freopen("connect.out", "w", stdout);
   int i,j,k,l,m,n,q,u,v;
   scanf("%d%d",&n,&q);
   comp=n;
   for(int i=1; i<=n; i++)
   {
      par[i]=i;
      siz[i]=1;
   }
   for(int i=1; i<=q; i++)
   {
      scanf("%s",str);
      if(str[0]=='?')
      {
         dig[i]=1;
         continue;
      }
      scanf("%d%d",&u,&v);
```

```
    if(u>v)
      swap(u,v);
    if(str[0]=='+')
    {
      pii ps = mp(u,v);
      strt[ps]=i;
      s.insert(ps);
      continue;
    }
    pii ps=mp(u,v);
    if(s.find(ps)==s.end())
      while(1);
    update(1,1,q,strt[ps],i-1,ps);
    s.erase(ps);
  }
  while(s.size())
  {
    it=s.begin();
    update(1,1,q,strt[*it],q,*it);
    pii gs=*it;
    s.erase(gs);
  }
  query(1,1,q);
  for(int i=1; i<=q; i++)
  {
    if(dig[i]==0)
      continue;
    printf("%d\n",ara[i]);
  }
}
```

44. K-D tree+ KNN (K-nearest neighbour)

```
//it can be viewed as multidimensional binary search tree
//dimension 0 based
//all distance are euclidian distance
#define dimension 3
#define lim 100010
struct co{
  LL x[dimension];
};
co arr[lim];
struct node{
  co now;
  //for left and right child
  int left;
  int right;
};
node bst[lim];
int axis;

bool comp(co p,co q)
{
  return p.x[axis]<q.x[axis]; //sort in terms of axis direction
}

//overall complexity n(logn)^2
void kdtree(co arr[],int st,int end,int depth,int &bstindex)
{
  if(st>end) return;
  axis=depth%dimension;
    //can be done in nlogn time by making optimizing here
```

```
   sort(arr+st,arr+end+1,comp);
      //debug_array(arr,9);
   int median=(st+end)/2;
   ++bstindex;
   int previndex=bstindex;
   bst[previndex].now=arr[median];

   if(median!=st) bst[previndex].left=bstindex+1;
   else bst[previndex].left=0;
   kdtree(arr,st,median-1,depth+1,bstindex);

   if(median!=end) bst[previndex].right=bstindex+1;
   else bst[previndex].right=0;
   kdtree(arr,median+1,end,depth+1,bstindex);
}

LL dist(co p,co q)  //taking square distance
{
   int i;
   LL ret=0;
   for(i=0;i<dimension;i++)
      ret+=(p.x[i]-q.x[i])*(p.x[i]-q.x[i]);
   return ret;
}

//normally k+logn complexity (not sure)
//kth nearest
void KNN(int bstnode,int bstindex,int depth,co query,int
k,priority_queue<LL> &Q)
{
   if(bstnode>bstindex) return;
   Q.push(dist(bst[bstnode].now,query));
   if(Q.size()>k) Q.pop();

   axis=depth%dimension;
   LL chc=bst[bstnode].now.x[axis]-query.x[axis];
   if(chc>=0) //go to left
   {
      KNN(bst[bstnode].left,bstindex,depth+1,query,k,Q);
      //special attention to > sign (sometimes >=)
      if(Q.top()>chc*chc || Q.size()<k)  //there is a chance of less
         KNN(bst[bstnode].right,bstindex,depth+1,query,k,Q);
      return;
   }

   //go to right
   KNN(bst[bstnode].right,bstindex,depth+1,query,k,Q);
   //special attention to > sign (sometimes >=)
   if(Q.top()>chc*chc || Q.size()<k)  //there is a chance of less
      KNN(bst[bstnode].left,bstindex,depth+1,query,k,Q);
   return;
}

//normally logn complexity
//number of points with sqrt(high) distance
void KNN2(int bstnode,int bstindex,int depth,co &query,LL
high,priority_queue<LL> &Q)
{
   if(!bstnode) return;
   //if(bstnode)
debug(bst[bstnode].now.x[0],bst[bstnode].now.x[1],bst[bstnode].now.x[
2]);
```

```
    if(bstnode>bstindex) return;
    Q.push(dist(bst[bstnode].now,query));
    if(Q.top()>=high) Q.pop();

    axis=depth%dimension;
    LL chc=bst[bstnode].now.x[axis]-query.x[axis];
    //print2(chc,query.x[0]);
    if(chc>=0) //go to left
    {
        KNN(bst[bstnode].left,bstindex,depth+1,query,high,Q);
        //special attention to > sign (sometimes >=)
        if(chc*chc<high)  //there is a chance of less
            KNN(bst[bstnode].right,bstindex,depth+1,query,high,Q);
        return;
    }

    //go to right
    KNN(bst[bstnode].right,bstindex,depth+1,query,high,Q);
    //special attention to > sign (sometimes >=)
    if(chc*chc<high)  //there is a chance of less
        KNN(bst[bstnode].left,bstindex,depth+1,query,high,Q);
    return;
}
int main(){
    //freopen("A.in","r",stdin);
    //freopen("A.out","w",stdout);
    int n,k;
    while(cin>>n>>k &&(n||k))
    {
        int i;
        for(int i = 1; i <= n; i++)
            scanf("%lld %lld %lld",&arr[i].x[0],&arr[i].x[1],&arr[i].x[2]);
        int bstindex=0;
        kdtree(arr,1,n,0,bstindex);
        int ans=0;
        for(i=1;i<=n;i++)
        {
            priority_queue<LL>q;
            KNN(1,bstindex,0,arr[i],(LL)k*(LL)k,q);
            ans+=q.size()-1;
        }
        ans/=2;
        printf("%d\n",ans);
    }
    return 0;
}
/*
Input :-
7 2
0 0 0
1 0 0
1 2 0
1 2 3
1000 1000 1000
1001 1001 1000
1001 999 1001
7 3
0 0 0
1 0 0
1 2 0
1 2 3
-1000 1000 -1000
```

```
-1001 1001 -1000
-1001 999 -1001
7 4
0 0 0
1 0 0
1 2 0
1 2 3
1000 -1000 1000
1001 -1001 1000
1001 -999 1001
Output :
3
6
9
*/
```

45. G-Driver Segment Tree

```
///straightly copied it from Grimoire

#include <bits/stdc++.h>
using namespace std;
#define ll long long
struct ext
{
   int mx,sec,cnt,tag;
   ll sum;
} tree[4000500];
int ara[2000000];

void update(int node)
{
   int lft=node<<1,rgt=node<<1|1;
   tree[node].sum=tree[node<<1].sum+tree[node<<1|1].sum;
   if(tree[lft].mx==tree[rgt].mx)
   {
      tree[node].mx=tree[lft].mx;
      tree[node].cnt=tree[lft].cnt+tree[rgt].cnt;
      tree[node].sec=max(tree[lft].sec,tree[rgt].sec);
      return ;
   }
   if(tree[lft].mx>tree[rgt].mx)
   {
      tree[node].mx=tree[lft].mx;
      tree[node].cnt=tree[lft].cnt;
      tree[node].sec=max(tree[lft].sec,tree[rgt].mx);
      return ;
   }
   swap(lft,rgt);
   tree[node].mx=tree[lft].mx;
   tree[node].cnt=tree[lft].cnt;
   tree[node].sec=max(tree[lft].sec,tree[rgt].mx);
   return ;
}
void build(int node,int st,int en)
{
   tree[node].tag=-1;
   tree[node].sum=0;
   if(st==en)
   {
      tree[node].mx=ara[st];
      tree[node].sec=-1;
      tree[node].cnt=1;
      tree[node].sum=ara[st];
      return ;
   }
```

```
    int mid=(st+en)>>1;
    build(node<<1,st,mid);
    build(node<<1|1,mid+1,en);
    update(node);
}
void modi(int node,int st,int en,int val)
{
    if(tree[node].mx<=val)
        return ;
    if(val>tree[node].sec)
    {
        tree[node].tag=val;
        tree[node].sum-=1LL*(tree[node].mx-val)*tree[node].cnt;
        tree[node].mx=val;
        return ;
    }
    int mid=(st+en)>>1;
    modi(node<<1,st,mid,val);
    modi(node<<1|1,mid+1,en,val);
    update(node);
}
void down(int node,int st,int en)
{
    if(tree[node].tag==-1)
        return ;
    int mid=(st+en)>>1;
    modi(node<<1,st,mid,tree[node].tag);
    modi(node<<1|1,mid+1,en,tree[node].tag);
    tree[node].tag=-1;
}
void modify(int node,int st,int en,int l,int r,int val)
{
    if(st>r or en<l)
        return ;
    if(st>=l and en<=r)
    {
        modi(node,st,en,val);
        return ;
    }
    int mid=(st+en)>>1;
    down(node,st,en);
    int lft=node<<1,rgt=lft|1;
    modify(lft,st,mid,l,r,val);
    modify(rgt,mid+1,en,l,r,val);
    update(node);
}
int querymx(int node,int st,int en,int l,int r)
{
    if(st>r or en<l)
        return 0;
    if(st>=l and en<=r)
        return tree[node].mx;
    int mid=(st+en)>>1;
    down(node,st,en);
    int p=querymx(node<<1,st,mid,l,r);
    int q=querymx(node<<1|1,mid+1,en,l,r);
    return max(p,q);
}
ll querysum(int node,int st,int en,int l,int r)
{
    if(st>r or en<l)
        return 0;
    if(st>=l && en<=r)
    {
        return  tree[node].sum;
    }
```

```
    down(node,st,en);
    int mid=(st+en)>>1;
    ll ret=0;
    ret+=querysum(node<<1,st,mid,l,r);
    ret+=querysum(node<<1|1,mid+1,en,l,r);
    return ret;
}
int main()
{
    int i,j,k,l,m,n,ty,ts;
    cin>>ts;
    while(ts--)
    {
        scanf("%d%d",&n,&m);
        for(int i=1; i<=n; i++)
            scanf("%d",&ara[i]);
        build(1,1,n);
        while(m--)
        {
            scanf("%d",&ty);
            if(ty==0)
            {
                scanf("%d%d%d",&l,&k,&j);
                modify(1,1,n,l,k,j);
                continue;
            }
            if(ty==1)
            {
                scanf("%d%d",&l,&k);
                j=querymx(1,1,n,l,k);
                printf("%d\n",j);
                continue ;
            }
            scanf("%d%d",&l,&k);
            ll ans=querysum(1,1,n,l,k);
            printf("%lld\n",ans);
            continue ;
        }
    }
    return 0;
}
```

46. Link Cut Tree

```
#include <bits/stdc++.h>
using namespace std;

struct Node
{   int sz, label; /* size, label */
    Node *p, *pp, *l, *r; /* parent, path-parent, left, right pointers */
    Node() { p = pp = l = r = 0; }
};

void update(Node *x)
{   x->sz = 1;
    if(x->l) x->sz += x->l->sz;
    if(x->r) x->sz += x->r->sz;
}

void rotr(Node *x)
{   Node *y, *z;
    y = x->p, z = y->p;
    if((y->l = x->r)) y->l->p = y;
    x->r = y, y->p = x;
    if((x->p = z))
    {   if(y == z->l) z->l = x;
```

```
      else z->r = x;
   }
   x->pp = y->pp;
   y->pp = 0;
   update(y);
}

void rotl(Node *x)
{  Node *y, *z;
   y = x->p, z = y->p;
   if((y->r = x->l)) y->r->p = y;
   x->l = y, y->p = x;
   if((x->p = z))
   {  if(y == z->l) z->l = x;
      else z->r = x;
   }
   x->pp = y->pp;
   y->pp = 0;
   update(y);
}

void splay(Node *x)
{  Node *y, *z;
   while(x->p)
   {  y = x->p;
      if(y->p == 0)
      {  if(x == y->l) rotr(x);
         else rotl(x);
      }
      else
      {  z = y->p;
         if(y == z->l)
         {  if(x == y->l) rotr(y), rotr(x);
            else rotl(x), rotr(x);
         }
         else
         {  if(x == y->r) rotl(y), rotl(x);
            else rotr(x), rotl(x);
         }
      }
   }
   update(x);
}

Node *access(Node *x)
{  splay(x);
   if(x->r)
   {  x->r->pp = x;
      x->r->p = 0;
      x->r = 0;
      update(x);
   }

   Node *last = x;
   while(x->pp)
   {  Node *y = x->pp;
      last = y;
      splay(y);
      if(y->r)
      {  y->r->pp = y;
         y->r->p = 0;
      }
      y->r = x;
      x->p = y;
      x->pp = 0;
      update(y);
```

```
      splay(x);
   }
   return last;
}

Node *root(Node *x)
{  access(x);
   while(x->l) x = x->l;
   splay(x);
   return x;
}

void cut(Node *x)
{  access(x);
   x->l->p = 0;
   x->l = 0;
   update(x);
}

void link(Node *x, Node *y)
{  access(x);
   access(y);
   x->l = y;
   y->p = x;
   update(x);
}

Node *lca(Node *x, Node *y)
{  access(x);
   return access(y);
}

int depth(Node *x)
{  access(x);
   return x->sz - 1;
}

class LinkCut
{  Node *x;

   public:
   LinkCut(int n)
   {  x = new Node[n];
      for(int i = 0; i < n; i++)
      {  x[i].label = i;
         update(&x[i]);
      }
   }

   virtual ~LinkCut()
   {  delete[] x;
   }

   void link(int u, int v)
   {  ::link(&x[u], &x[v]);
   }

   void cut(int u)
   {  ::cut(&x[u]);
   }

   int root(int u)
   {  return ::root(&x[u])->label;
   }

   int depth(int u)
```

```
    {   return ::depth(&x[u]);
    }

    int lca(int u, int v)
    {   return ::lca(&x[u], &x[v])->label;
    }
};

int main(void)
{
    int i,j,k,l,m,n;
    scanf("%d%d",&n,&m);
    char str[50];
    LinkCut *vox= new LinkCut(n+1);;
    while(m--)
    {
        scanf("%s%d",str,&l);
        if(str[0]=='c')
        {
            vox->cut(l);
            continue;
        }
        scanf("%d",&k);
        if(str[1]=='c')
        {
            int f=vox->lca(l,k);
            printf("%d\n",f);
            continue;
        }
        vox->link(l,k);
    }
    return 0;
}
```

47. Link Cut Tree 2

```
#include <bits/stdc++.h>
using namespace std;
//to make a node root of its tree just access it and then take its left
parent tree[v].l
//and make its pp=v and p=0 and tree[tree[v].l].rev^=1;
struct Node
{
    int rev;
    int realval,calval;
    int sz, label; /* size, label */
    int p, pp, l, r; /* parent, path-parent, left, right pointers */
};
Node tree[MXN];
pii edge[MXN];
void update(int x)
{
    tree[x].sz = 1;
    tree[x].sz += tree[tree[x].l].sz;
    tree[x].sz+=tree[tree[x].r].sz;

tree[x].calval=min(min(tree[x].realval,tree[tree[x].l].calval),tree[tree[x].r].
calval);
}
void rotr(int x)
{
    int y, z;
    y = tree[x].p, z = tree[y].p;
    if((tree[y].l = tree[x].r))
        tree[tree[y].l].p = y;
    tree[x].r=y, tree[y].p= x;
    if((tree[x].p = z))
    {
```

```
    if(y == tree[z].l)
      tree[z].l = x;
    else
      tree[z].r = x;
  }
  tree[x].pp = tree[y].pp;
  tree[y].pp=0;
  update(y);
}
void rotl(int x)
{
  int y,z;
  y = tree[x].p, z = tree[y].p;
  if((tree[y].r = tree[x].l))
    tree[tree[y].r].p = y;
  tree[x].l=y, tree[y].p = x;
  if((tree[x].p = z))
  {
    if(y == tree[z].l)
      tree[z].l=x;
    else
      tree[z].r=x;
  }
  tree[x].pp = tree[y].pp;
  tree[y].pp = 0;
  update(y);
}
void up(int x)
{
  if(tree[x].p)
    up(tree[x].p);
  if(tree[x].rev)
  {
    tree[x].rev=0;
    swap(tree[x].l,tree[x].r);
    tree[tree[x].l].rev^=1;
    tree[tree[x].r].rev^=1;
  }
}
void splay(int x)
{
  int y, z;
  up(x);
  while(tree[x].p)
  {
    y = tree[x].p;
    if(tree[y].p == 0)
    {
      if(x == tree[y].l)
        rotr(x);
      else
        rotl(x);
    }
    else
    {
      z = tree[y].p;
      if(y == tree[z].l)
      {
        if(x == tree[y].l)
          rotr(y), rotr(x);
        else
          rotl(x), rotr(x);
      }
      else
      {
        if(x == tree[y].r)
```

```
                rotl(y), rotl(x);
            else
                rotr(x), rotl(x);
        }
    }
  }
  update(x);
}
int access(int x)
{
  splay(x);
  if(tree[x].r)
  {
    tree[tree[x].r].pp = x;
    tree[tree[x].r].p = 0;
    tree[x].r = 0;
    update(x);
  }

  int last = x;
  while(tree[x].pp)
  {
    int y = tree[x].pp;
    last = y;
    splay(y);
    if(tree[y].r)
    {
      tree[tree[y].r].pp = y;
      tree[tree[y].r].p = 0;
    }
    tree[y].r=x;
    tree[x].p=y;
    tree[x].pp = 0;
    update(y);
    splay(x);
  }
  return last;
}
int findroot(int x)
{
  access(x);
  while(tree[x].l)
    x = tree[x].l;
  splay(x);
  return x;
}
int findoime(int x)
{
  if(x==0)
    return 0;
  //if(tree[x].sz==1)return 0;
  int v=x;
  while(1)
  {
    if(tree[v].r==0)
      break;
    v=tree[v].r;
  }
  splay(v);
  return v;
}
int cut(int x,int ds=0)
{
  access(x);
  int gs=0;
  int v=tree[x].l;
```

```
    tree[tree[x].l].p = 0;
    tree[x].l = 0;
    // tree[v].pp=tree[x].pp;
    //tree[x].pp=0;
    update(x);
    if(ds)
    {

        gs=v;
    }
//update(x);
    return gs;
}
void link(int x, int y)
{
    access(x);
    access(y);
    assert(tree[x].pp==0&&tree[y].pp==0);
    if(tree[y].l)
    {
        tree[tree[y].l].pp=y;
        tree[tree[y].l].p=0;
        tree[tree[y].l].rev^=1;
        tree[y].l=0;
        update(y);
    }
    tree[y].l=x;
    tree[x].p=y;
    tree[x].pp=0;
    update(y);
}
int lca(int x, int y)
{
    access(x);
    return access(y);
}
int depth(int x)
{
    access(x);
    return tree[x].sz - 1;
}
int P[MXN];
int findo(int v)
{
    if(v==P[v])
        return v;
    return P[v]=findo(P[v]);
}
int tree2[1000000];
void update(int node,int st,int en,int l,int r)
{
    if(st>l or en<l)
        return ;
    if(st==en)
    {
        tree2[node]+=r;
        return ;
    }
    int mid=(st+en)>>1;
    update(node*2,st,mid,l,r);
    update(node*2+1,mid+1,en,l,r);
    tree2[node]=tree2[node*2]+tree2[node*2+1];
}
int query(int node,int st,int en,int l,int r)
{
    if(st>r or en<l or l>r)
```

```
        return 0;
    if(st>=l and en<=r)
        return tree2[node];
    int mid=(st+en)>>1;
    int v=query(node*2,st,mid,l,r);
    int w=query(node*2+1,mid+1,en,l,r);
    return v+w;

}
void build(int node,int st,int en)
{
    tree2[node]=0;
    if(st==en)
    {
        return ;
    }
    int mid=(st+en)>>1;
    build(node*2,st,mid);
    build(node*2+1,mid+1,en);

}
vector<pii>vc[MXN];
int ou[MXN];
int main()
{
//freopen ("inp.txt","r",stdin);
    int i,j,k,l,m,n,Q,ts;
    cin>>ts;
    while(ts--)
    {
        scanf("%d%d",&n,&m);
        scanf("%d",&Q);
        for(int i=1; i<=m; i++)
            vc[i].clear();
        build(1,1,m);
        //  int i,j,k,l,m,n;
        //  scanf("%d%d",&n,&m);
        //  int compo=0//;
        for(int i=0; i<=n+m; i++)
        {
            tree[i].l=tree[i].r=tree[i].p=tree[i].pp=0;
            tree[i].sz=1;
            tree[i].realval=tree[i].calval=INT_MAX;
            if(i>n)
            {
                //   cout<<i<<endl;
                tree[i].realval=tree[i].calval=i;
            }
            P[i]=i;
            tree[i].rev=0;
        }
//    cout<<tree[4].realval<<endl;
        int comp=n;
        //,Q int Q;
        for(int i=1; i<=m; i++)
        {
            scanf("%d%d",&l,&k);
            edge[i]=mp(l,k);
        }
        //  cout<<"sdf"<<endl;

        for(int i=1; i<=Q; i++)
        {
            scanf("%d%d",&l,&k);
            vc[k].pb(mp(l,i));
            // edge[i]=mp(l,k);
```

```
    }
    // cout<<"df"<<endl;
    for(int i=1; i<=m; i++)
    {
       //// scanf("%d%d",&l,&k);
       l=edge[i].fs,k=edge[i].sc;

       int v=findo(l);
       int w=findo(k);
       if(v!=w)
       {
          comp--;
          //   cout<<comp<<" "<<i<<endl;
          P[v]=w;
          // tree[i+n].realval=i;
          //    cout<<tree[i+n].realval<<" df "<<i<<" "<<(i+n)<<"
"<<n<<endl;
          link(l,i+n);
          link(i+n,k);

          //continue;
       }
       else if(l!=k)
       {
          int lc=lca(l,k);
          //  cout<<lc<<endl;

          //access(lc);
          //int ab=findoime(lc);
          int ab=cut(lc,1);
          if(lc>n)
          {
             //  if(ab!=0)while(1);
          }
       //  if(ab!=0&&ab<=n)while(1);
       //  cout<<ab<<endl;
       access(l);
       int w=tree[l].sz;
       int f=tree[l].calval;
       access(k);
       w+=tree[k].sz;
       f=min(tree[k].calval,f);
       assert(w>3);
       //   cout<<"ff "<<f<<" "<<lc<<" "<<ab<<endl;
       //  if(f==INT_MAX)while(1);

       update(1,1,m,f-n,-1);
       int a=edge[f-n].fs;
       int b=edge[f-n].sc;
       //  cout<<f<<" fd "<<i<<" "<<lc<<" "<<ab<<" "<<a<<" "<<b<<endl;
       int lc2=lca(a,b);
       if(lc2==a)
       {
          cut(f);
          cut(b);
       }
       else if(lc2==b)
       {
          cut(f);
          cut(a);
       }
       else
       {
          if(lc2<=n)
             while(1);
       }
```

```
            // else while(1);
            if(ab)
            {
               //    link(ab,lc);
               access(lc);
               tree[lc].l=ab;
               tree[ab].p=lc;
               update(lc);
            }
            link(l,i+n);
            link(i+n,k);
         }
         //   cout<<i<<endl;
         // edge[i]=mp(l,k);
         if(l!=k)
            update(1,1,m,i,1);
         // cout<<"ds"<<endl;
         for(int j=0; j<vc[i].size(); j++)
         {
            int l=vc[i][j].fs;
            int f=query(1,1,m,1,l-1);
            //  cout<<f<<" "<<l<<" "<<comp<<" "<<i<<endl;
            ou[vc[i][j].sc]=comp+f;
         }
      }
      // cout<<"fdssfd"<<endl;
      for(int i=1; i<=Q; i++)
      {
         printf("%d\n",ou[i]);

      }
   }
   return 0;
}
```

48. Link Cut Tree 3

```
/*
getSum(u) := returns sum of values in nodes in u's subtree.
add(u, x) := adds x to values of every node in u's subtree (including u).
swap(u, v) := swaps whole subtrees of u and v if and only if the subtrees
are */
//here second second splay tree show us how to use lazy propagation
//first splay tree show us how to tell the number of nodes under its
subtree

struct Node
{
   ll lazy;
   int sz;
   //int sz, label; /* size, label */
   int p, pp, l, r; /* parent, path-parent, left, right pointers */
};
Node tree[MXN];
int P[MXN],val[MXN],col[MXN];

void update2(int x)
{
   // here you can update anything about splay tree1 er information
   // look lct1 for clearance
}
void rotr(int x)
{
   int y, z;
   y = tree[x].p, z = tree[y].p;
   if((tree[y].l = tree[x].r))
      tree[tree[y].l].p = y;
```

```
    tree[x].r=y, tree[y].p= x;
    if((tree[x].p = z))
    {
        if(y == tree[z].l)
            tree[z].l = x;
        else
            tree[z].r = x;
    }
    tree[x].pp = tree[y].pp;
    tree[y].pp=0;
    update2(y);
}

void rotl(int x)
{
    int y,z;
    y = tree[x].p, z = tree[y].p;
    if((tree[y].r = tree[x].l))
        tree[tree[y].r].p = y;
    tree[x].l=y, tree[y].p = x;
    if((tree[x].p = z))
    {
        if(y == tree[z].l)
            tree[z].l=x;
        else
            tree[z].r=x;
    }
    tree[x].pp = tree[y].pp;
    tree[y].pp = 0;
    update2(y);
}
void up(int x)
{
    if(tree[x].p)
        up(tree[x].p);
    if(tree[x].lazy)
    {
        tree[x].sz+=tree[x].lazy;
        tree[tree[x].l].lazy+=tree[x].lazy;
        tree[tree[x].r].lazy+=tree[x].lazy;
        tree[x].lazy=0;
    }
    update2(x);
}
void splay(int x)
{
    int y, z;
    up(x);
    while(tree[x].p)
    {
        y = tree[x].p;
        if(tree[y].p == 0)
        {
            if(x == tree[y].l)
                rotr(x);
            else
                rotl(x);
        }
        else
        {
            z = tree[y].p;
            if(y == tree[z].l)
            {
                if(x == tree[y].l)
                    rotr(y), rotr(x);
                else
```

```
                rotl(x), rotr(x);
            }
            else
            {
                if(x == tree[y].r)
                    rotl(y), rotl(x);
                else
                    rotr(x), rotl(x);
            }
        }
    }
    update2(x);
}

int access(int x)
{
    splay(x);
    if(tree[x].r)
    {
        tree[tree[x].r].pp = x;
        tree[tree[x].r].p = 0;
        tree[x].r = 0;
        update2(x);
    }

    int last = x;
    while(tree[x].pp)
    {
        int y = tree[x].pp;
        last = y;
        splay(y);
        if(tree[y].r)
        {
            tree[tree[y].r].pp = y;
            tree[tree[y].r].p = 0;
        }
        tree[y].r=x;
        tree[x].p=y;
        tree[x].pp = 0;
        update2(y);
        splay(x);
    }
    return last;
}
int cut(int x,int ds=0)
{
    // cout<<x<<endl;
    access(x);
    int gs=0;
    int v=tree[x].l;
    // if(x==5)cout<<tree[x].sz<<" fdf"<<endl;
    tree[tree[x].l].p = 0;
    tree[x].l = 0;
    tree[v].lazy-=tree[x].sz;
    update2(x);
}

void link(int x, int y)
{
    if(x==0)
        return ;
    access(x);
    access(y);
    tree[y].l=x;
    tree[x].p=y;
    tree[x].lazy+=tree[y].sz;
```

```
    update2(y);
}

int lca(int x, int y)
{
    access(x);
    return access(y);
}
vector<int>vc[MXN];
struct node
{
    int l,r,p,sz;
    ll val,sum,add;
};
node tree2[400000];
void update(int v)
{
    tree2[v].sz=tree2[tree2[v].l].sz+tree2[tree2[v].r].sz+1;
    tree2[v].val=tree2[v].val+tree2[v].add;
    tree2[v].sum=tree2[v].val;
    tree2[v].add=0;

tree2[v].sum=tree2[tree2[v].l].sum+tree2[tree2[v].r].sum+tree2[v].sum;
}
void push(int v)
{
    if(tree2[v].add)
    {
        tree2[tree2[v].l].add+=tree2[v].add;
        tree2[tree2[v].l].sum+=tree2[tree2[v].l].sz*tree2[v].add;
        tree2[tree2[v].r].add+=tree2[v].add;
        tree2[tree2[v].r].sum+=tree2[tree2[v].r].sz*tree2[v].add;
    }
    update(v);
}
void Zig(int c)
{
    int p=tree2[c].p,q=tree2[p].p;
    tree2[p].l=tree2[c].r;
    if(tree2[p].l!=0)
        tree2[tree2[p].l].p=p;
    tree2[c].r=p,tree2[c].p=q,tree2[p].p=c;
    if(q!=0)
    {
        if(tree2[q].l==p)
            tree2[q].l=c;
        else
            tree2[q].r=c;
    }
    update(p);
}
void Zag(int c)
{
    int p=tree2[c].p,q=tree2[p].p;
    tree2[p].r=tree2[c].l;
    if(tree2[p].r!=0)
        tree2[tree2[p].r].p=p;
    tree2[c].l=p,tree2[c].p=q,tree2[p].p=c;
    if(q!=0)
    {
        if(tree2[q].l==p)
            tree2[q].l=c;
        else
            tree2[q].r=c;
    }
    update(p);
```

```
}
void pull(int v)
{
   if(v==0)
      return ;
   pull(tree2[v].p);
   push(v);
}
int root;
void splay2(int c)
{
   pull(c);
   while(tree2[c].p!=0)
   {
      int p=tree2[c].p,q=tree2[p].p;
      if(q!=0&&tree2[q].l==p)
      {
         if(tree2[p].l==c)
            Zig(p),Zig(c);
         else
            Zag(c),Zig(c);
      }
      else if(q&&tree2[q].r==p)
      {
         if(tree2[p].r==c)
            Zag(p),Zag(c);
         else
            Zig(c),Zag(c);
      }
      else
      {
         if(tree2[p].l==c)
            Zig(c);
         else
            Zag(c);
      }
   }
//   cout<<c<<"fdsf"<<endl;
   root=c;
   update(c);
}
int srch(int tomo)
{
   int c=root;
   // cout<<tomo<<" "<<c<<endl;
   while(c)
   {
      push(c);
      if(tree2[tree2[c].l].sz>=tomo)
         c=tree2[c].l;
      else
      {
         tomo-=tree2[tree2[c].l].sz+1;
         if(tomo==0)
         {
            //    cout<<c<<" sd"<<endl;
            splay2(c);
            return c;
         }
         c=tree2[c].r;
      }
      //  cout<<c<<endl;
   }
}
int last;
void dfs(int v,int p)
```

```
{
   if(v!=1)
   {
      tree2[last].r=v;
      tree2[v].p=last;
      update(last);
      splay2(v);
   }
   last=v;
   for(int i=0; i<vc[v].size(); i++)
   {

      int w=vc[v][i];
      if(w==p)
         continue;
      dfs(w,v);
   }
   if(p)
      link(p,v);
   P[v]=p;
}
int find_v(int v)
{
   int ager=0,ini=v;
   int ans=tree2[tree2[v].l].sz+1;
   ager=v;
   v=tree2[v].p;

   while(v)
   {
      if(tree2[v].r==ager)
         ans+=tree2[tree2[v].l].sz+1;
      ager=v;
      v=tree2[v].p;
   }
   splay2(ini);
   return (ans);
}
ll solve(int v)
{
   int pos=find_v(v);
   access(v);
   int sz=tree[v].sz;
   //cout<<sz<<endl;
   if(pos==1)
   {
      if(tree2[root].sz==sz)
      {
         return tree2[root].sum;
      }
      assert(0);
   }
   srch(pos-1);
   int tmp=root;
   if(tree2[tree2[root].r].sz==sz)
   {
      return tree2[tree2[root].r].sum;
   }
   root=tree2[tmp].r;
   tree2[root].p=0;
   srch(sz+1);
   ll re=tree2[tree2[root].l].sum;
   tree2[tmp].r=root;
   tree2[root].p=tmp;
   root=tmp;
   return re;
```

```
}
void add_v(int v,ll d)
{
   tree2[v].add+=d;
   tree2[v].sum+=(d*tree2[v].sz);

}
void solve2(int v,int ad)
{
   int pos=find_v(v);
   access(v);
   int sz=tree[v].sz;
   if(pos==1)
   {
      if(tree2[root].sz==sz)
      {
         add_v(root,ad);
         return ;
      }
      assert(0);
      return ;
   }
   srch(pos-1);
   int tmp=root;
   if(tree2[tree2[root].r].sz==sz)
   {
      add_v(tree2[root].r,ad);
      update(root);
      return ;
   }
   root=tree2[tmp].r;
   tree2[root].p=0;
   srch(sz+1);
   add_v(tree2[root].l,ad);
   update(root);
   tree2[tmp].r=root;
   tree2[root].p=tmp;
   root=tmp;
   update(tmp);
   return ;
}
void solve3(int u,int v)
{
   int lc=lca(u,v);
   if(lc==u||lc==v)
   {
      printf("-1\n");
      return ;
   }
   int pos=find_v(u);
   access(u);
   int szu=tree[u].sz;
   srch(pos-1);
   int root1,root2,tp;
   if(tree2[tree2[root].r].sz==szu)
   {
      root1=tree2[root].r;
      tree2[root1].p=0;
      tree2[root].r=0;
      update(root);
   }
   else
   {
      tp=root;
      root=tree2[root].r;
      tree2[root].p=0;
```

```
    srch(szu+1);
    root1=tree2[root].l;
    tree2[root1].p=0;
    tree2[root].l=0;
    update(root);
    tree2[root].p=tp;
    tree2[tp].r=root;
    update(tp);
    root=tp;
}
cut(u);
int pos2=find_v(v);
access(v);
int szv=tree[v].sz;
srch(pos2-1);
if(tree2[tree2[root].r].sz==szv)
{
    root2=tree2[root].r;
    tree2[root2].p=0;
    tree2[root].r=0;
    update(root);
}
else
{
    tp=root;
    root=tree2[root].r;
    tree2[root].p=0;
    srch(szv+1);
    root2=tree2[root].l;
    tree2[root2].p=0;
    tree2[root].l=0;
    update(root);
    tree2[root].p=tp;
    tree2[tp].r=root;
    root=tp;
    update(root);
}
int a=P[u],b=P[v];
cut(v);
int pos3=find_v(P[u]);
srch(pos3);
int rt=tree2[root].r;
tree2[root].r=0;
update(root);
tp=root;
root=root2;
srch(szv);
tree2[root].r=rt;
tree2[rt].p=root;
update(root);
tree2[tp].r=root;
tree2[root].p=tp;
root=tp;
update(root);
P[v]=a;
link(P[v],v);
int pos4=find_v(b);
srch(pos4);
rt=tree2[root].r;
tree2[root].r=0;
update(root);
tp=root;
root=root1;
srch(szu);
tree2[root].r=rt;
tree2[rt].p=root;
```

```
    update(root);
    tree2[tp].r=root;
    tree2[root].p=tp;
    root=tp;
    update(root);
    P[u]=b;
    link(P[u],u);
}
int main()
{
    int  i,j,k,l,m,n,q;
    cin>>n>>q;
    root=1;
    for(int i=1; i<=n; i++)
    {
        scanf("%d",&val[i]);
        tree[i].sz=1;
        tree[i].p=tree[i].l=tree[i].r=tree[i].pp=0;
        tree[i].lazy=0;
        tree2[i].sum=tree2[i].val=val[i];
        tree2[i].add=0;
        tree2[i].sz=1;
        tree2[i].l=tree2[i].r=tree2[i].p=0;
    }
    for(int i=1; i<n; i++)
    {
        scanf("%d%d",&l,&k);
        vc[l].pb(k);
        vc[k].pb(l);
    }
    dfs(1,0);
    while(q--)
    {
        int ty;
        scanf("%d",&ty);
        if(ty==1)
        {
            scanf("%d",&l);
            ll ans=solve(l);
            printf("%lld\n",ans);
            continue;
        }
        if(ty==2)
        {
            scanf("%d%d",&l,&k);
            solve2(l,k);
            continue;
        }
        scanf("%d%d",&l,&k);
        solve3(l,k);
    }
}
```

49. Wavelet Tree

```
#include<bits/stdc++.h>
using namespace std;
//#include "bitmap.hpp"
using namespace std;
typedef vector<int>::iterator iter;

//Wavelet tree with succinct representation of bitmaps
struct WaveTreeSucc
{
    vector<vector<int> > C;
    int s;
```

```
  // sigma = size of the alphabet, ie., one more than the maximum
element
  // in S.
  WaveTreeSucc(vector<int> &A, int sigma) : C(sigma*2), s(sigma)
  {
    build(A.begin(), A.end(), 0, s-1, 1);
  }

  void build(iter b, iter e, int L, int U, int u)
  {
    if (L == U)
      return;
    int M = (L+U)/2;

    // C[u][i] contains number of zeros until position i-1: [0,i)
    C[u].reserve(e-b+1);
    C[u].push_back(0);
    for (iter it = b; it != e; ++it)
      C[u].push_back(C[u].back() + (*it<=M));

    iter p = stable_partition(b, e, [=](int i)
    {
      return i<=M;
    });

    build(b, p, L, M, u*2);
    build(p, e, M+1, U, u*2+1);
  }

  // Count occurrences of number c until position i.
  // ie, occurrences of c in positions [i,j]
  int rank(int c, int i) const
  {
    // Internally we consider an interval open on the left: [0, i)
    i++;
    int L = 0, U = s-1, u = 1, M, r;
    while (L != U)
    {
      M = (L+U)/2;
      r = C[u][i];
      u*=2;
      if (c <= M)
        i = r, U = M;
      else
        i -= r, L = M+1, ++u;
    }
    return i;
  }

  // Find the k-th smallest element in positions [i,j].
  // The smallest element is k=1
  int quantile(int k, int i, int j) const
  {
    // internally we we consider an interval open on the left:  [i, j)
    j++;
    int L = 0, U = s-1, u = 1, M, ri, rj;
    while (L != U)
    {
      M = (L+U)/2;
      ri = C[u][i];
      rj = C[u][j];
      u*=2;
      if (k <= rj-ri)
        i = ri, j = rj, U = M;
      else
        k -= rj-ri, i -= ri, j -= rj,
```

```
                L = M+1, ++u;
        }
        return U;
    }

    // Count number of occurrences of numbers in the range [a, b]
    // present in the sequence in positions [i, j], ie, if representing a grid it
    // counts number of points in the specified rectangle.
    mutable int L, U;
    int range(int i, int j, int a, int b) const
    {
        if (b < a or j < i)
            return 0;
        L = a;
        U = b;
        return range(i, j+1, 0, s-1, 1);
    }

    int range(int i, int j, int a, int b, int u) const
    {
        if (b < L or U < a)
            return 0;
        if (L <= a and b <= U)
            return j-i;
        int M = (a+b)/2, ri = C[u][i], rj = C[u][j];
        return range(ri, rj, a, M, u*2) +
               range(i-ri, j-rj, M+1, b, u*2+1);
    }
};

int arr[100005];
int main()
{
    int n, m;
    scanf("%d %d", &n, &m);
    vector<int> ara(n);
    set<int> st;
    map<int, int> mp;
    for(int i = 0; i<n; i++)
    {
        scanf("%d", &ara[i]);
        st.insert(ara[i]);
    }
    int pos = 1;
    for(int x : st)
        mp[x] = pos++, arr[pos-1] = x;
    for(int i = 0; i<n; i++)
        ara[i] = mp[ara[i]];

    WaveTreeSucc sol(ara, pos);

    for(int i = 0; i<m; i++)
    {
        int l, r, k;
        scanf("%d %d %d", &l, &r, &k);
        l--, r--;
        printf("%d\n", arr[sol.quantile(k, l, r)]);
    }
    return 0;
}
```

50. Splay Tree

```
#include <bits/stdc++.h>
using namespace std;

struct node
```

```
{
   int v,l,r,p,sz,lzy;
};
int n,root;
node tree[400000];
void update(int c)
{
   tree[c].sz=tree[tree[c].l].sz+tree[tree[c].r].sz+1;

}
void init()
{
   n=0,root=0;
   tree[0].sz=0;
   tree[0].l=0;
   tree[0].r=0;
   tree[0].lzy=0;
   tree[0].p=0;
}
int newnode(int v,int par)
{
   n++;
   tree[n].v=v;
   tree[n].l=tree[n].r=0;
   tree[n].lzy=0;
   tree[n].p=par;
   return n;
}
void Zig(int c)
{
   int p=tree[c].p,q=tree[p].p;
   tree[p].l=tree[c].r;
   if(tree[p].l!=0)
      tree[tree[p].l].p=p;
   tree[c].r=p,tree[c].p=q,tree[p].p=c;
   if(q!=0)
   {
      if(tree[q].l==p)
         tree[q].l=c;
      else
         tree[q].r=c;
   }
   update(p);
}
void Zag(int c)
{
   int p=tree[c].p,q=tree[p].p;
   tree[p].r=tree[c].l;
   if(tree[p].r!=0)
      tree[tree[p].r].p=p;
   tree[c].l=p,tree[c].p=q,tree[p].p=c;
   if(q!=0)
   {
      if(tree[q].l==p)
         tree[q].l=c;
      else
         tree[q].r=c;
   }
   update(p);
}
void splay(int c)
{
```

```
  while(tree[c].p!=0)
  {
    int p=tree[c].p,q=tree[p].p;
    if(q!=0&&tree[q].l==p)
    {
      if(tree[p].l==c)
        Zig(p),Zig(c);
      else
        Zag(c),Zig(c);
    }
    else if(q&&tree[q].r==p)
    {
      if(tree[p].r==c)
        Zag(p),Zag(c);
      else
        Zig(c),Zag(c);
    }
    else
    {
      if(tree[p].l==c)
        Zig(c);
      else
        Zag(c);
    }
  }
  root=c;
  update(c);
}
int srch(int tomo)
{
  int c=root;
  while(c)
  {
    if(tree[c].lzy==1)
    {
      tree[c].lzy=0;
      swap(tree[c].l,tree[c].r);
      tree[tree[c].l].lzy^=1;
      tree[tree[c].r].lzy^=1;
    }
    if(tree[tree[c].l].sz>=tomo)
      c=tree[c].l;
    else
    {
      tomo-=tree[tree[c].l].sz+1;
      if(tomo==0)
      {
        splay(c);
        return c;
      }
      c=tree[c].r;

    }
  }
}
void rotatee (int l,int r)
{
  if(r-l+1==tree[root].sz)
  {
    tree[root].lzy^=1;
    return ;
  }
```

```
    if(l==1)
    {
        srch(r+1);
        tree[tree[root].l].lzy^=1;
        return ;
    }
    if(r==tree[root].sz)
    {
        srch(l-1);
        tree[tree[root].r].lzy^=1;
        return ;
    }
    srch(l-1);
    int tmp=root;
    root=tree[root].r;
    tree[root].p=0;
    srch(r-tree[tree[tmp].l].sz);
    tree[tmp].r=root;
    tree[root].p=tmp;
    root=tmp;
    update(root);
    tree[tree[tree[root].r].l].lzy^=1;
}
void cut(int l,int r,int m)
{
    if(r-l+1==tree[root].sz)
    {
        return ;
    }
    int tmp;
    if(l==1)
    {
        srch(r+1);
        tmp=tree[root].l;
        tree[root].l=0;
    }
    else if(r==tree[root].sz)
    {
        srch(l-1);
        tmp=tree[root].r;
        tree[root].r=0;
    }
    else
    {
        srch(l-1);
        tmp=root;
        root=tree[root].r;
        tree[root].p=0;
        srch(r-tree[tree[tmp].l].sz);
        tree[tmp].r=root;
        tree[root].p=tmp;
        root=tmp;
        tmp=tree[tree[root].r].l;
        tree[tree[root].r].l=0;
        update(tree[root].r);
    }
    tree[tmp].p=0;
    update(root);
    if(m==0)
    {
        srch(1);
        tree[root].l=tmp;
```

```
        tree[tmp].p=root;
    }
    else if(m==tree[root].sz)
    {
        srch(m);
        tree[root].r=tmp;
        tree[tmp].p=root;
    }
    else
    {
        srch(m);
        int t1=root;
        root=tree[root].r;
        tree[root].p=0;
        srch(1);
        tree[t1].r=root;
        tree[root].p=t1;
        swap(t1,root);
        tree[t1].l=tmp;
        tree[tmp].p=t1;
        update(t1);
    }
    update(root);

}
int main()
{
    int i,j,k,l,M,N;
    while(scanf("%d%d",&N,&M))
    {
        if(N==-1||M==-1)
            break;
        init();
        for(int i=1; i<=N; i++)
        {
            l=newnode(i,i-1);
            // cout<<i<<" "<<l<<" "<<n<<endl;
            tree[i-1].r=l;
            splay(l);

        }
        // cout<<tree[n].sz<<" "<<l<<" "<<n<<endl;
        while(M--)
        {
            char str[90];
            scanf("%s%d%d",str,&l,&k);
            if(str[0]=='C')
            {
                scanf("%d",&j);
                cut(l,k,j);
                continue;
            }
            rotatee(l,k);
        }
        for(int i=1; i<=n; i++)
        {
            int l=srch(i);
            if(i>1)
                printf(" ");
            printf("%d",tree[l].v);
        }
        printf("\n");
```

```
    }
}
```

51. Rectange Union (Jubair)

```
#define ll long long
#define pb push_back
struct node
{
    int x1,x2,y,type;
    bool operator < (const node & p) const
    {
        return y<p.y;
    }
};
ll tree[700000],str[700000];
int lazy[700000];
vector<node>ac;
vector<int>vc;
void update(int node,int st,int en,int b,int e,int val)
{
    if(st>e or en<b)
        return ;
    if(st>=b and en<=e)
    {
        lazy[node]+=val;
        //    if(b==1&&e==1)cout<<lazy[node]<<" "<<vc[en+1]-vc[st]<<"
"<<vc[en+1]<<" "<<vc[st]<<" "<<en<<" "<<st<<endl;
        if(lazy[node])
        {
            tree[node]=vc[en+1]-vc[st];
        }
        else
        {
            tree[node]=str[node];
        }
        return ;
    }
    int mid=(st+en)/2;
    update(node*2,st,mid,b,e,val);
    update(node*2+1,mid+1,en,b,e,val);
    str[node]=tree[node*2]+tree[node*2+1];
    if(lazy[node])
    {
        tree[node]=vc[en+1]-vc[st];
    }
    else
    {
        tree[node]=str[node];
    }
}
map<int,int>name;
int main()
{
    int i,j,k,l,m,n,x1,x2,y1,y2,cnt,test,casio=1;
    scanf("%d",&test);
    while(test--)
    {
        ll f,d,e,ans=0;
        scanf("%d",&n);
        ans=0;
        vector<int>bit;
        bit.clear();
        ac.clear();
        vc.clear();
        name.clear();
        memset(tree,0,sizeof tree);
```

```
memset(lazy,0,sizeof lazy);
memset(str,0,sizeof str);
node gr;
while(n--)
{
   scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
   bit.pb(x1);
   bit.pb(x2);
   gr.x1=x1;
   gr.x2=x2;
   gr.y=y1;
   gr.type=1;
   ac.pb(gr);
   gr.y=y2;
   gr.type=-1;
   ac.pb(gr);
}
sort(bit.begin(),bit.end());
sort(ac.begin(),ac.end());
name[bit[0]]=0;
vc.pb(bit[0]);
cnt=0;
for(i=1; i<bit.size(); i++)
{
   if(bit[i]==bit[i-1])
      continue;
   vc.pb(bit[i]);
   name[bit[i]]=++cnt;
}

f=ac[0].y;
d=name[ac[0].x1];
e=name[ac[0].x2]-1;
      update(1,0,cnt,d,e,1);
      // cout<<d<<" "<<e<<endl;
      //   cout<<tree[1]<<endl;
      for(i=1; i<ac.size(); i++)
      {
         d=(ac[i].y-f);
         ans+=(d*(tree[1]));
         //  cout<<d<<" "<<tree[1]<<" "<<d*tree[1]<<endl;
         d=name[ac[i].x1];
         e=name[ac[i].x2]-1;
         //   cout<<d<<" "<<e<<endl;
         update(1,0,cnt,d,e,ac[i].type);
         // cout<<tree[1]<<endl;
         f=ac[i].y;
      }
      printf("Case %d: %lld\n",casio++,ans);
   }
}
```

52. Rectangle Union Without Compress

```
/*    for initialize tree set :
      tree[1] = 0 ;
      prop[1] = -1 ;          */
long long tree[mx_coordinate*4], prop[mx_coordinate*4];

void relax(int nd,int l,int r) {
   if(l != r && prop[nd] == -1) {
      int lc = 2*nd, rc = lc + 1 ;
      tree[nd] = 0 ;
      tree[lc] = 0 ;
      tree[rc] = 0 ;
      prop[lc] = -1 ;
```

```
      prop[rc] = -1 ;
   }
}
void update(int nd,int l,int r,int ql,int qr,int v) {
   if(r<ql || l>qr)
      return ;
   relax(nd,l,r);
   int lc = 2*nd, rc = lc + 1 ;

   if(l>=ql && r<=qr) {
      prop[nd] = max(0LL,prop[nd]) + v ;
//printf("%d %d %lld\n",l,r,prop[nd]);
      if(prop[nd]>0)
         tree[nd] = r-l+1 ;
      else {
         if(l != r) {
            prop[nd] = (prop[lc] == -1 && prop[rc] == -1 ? -1 : 0) ;
            tree[nd] = (prop[nd] == -1 ? 0 : tree[lc] + tree[rc]);
         }
         else {
            prop[nd] = -1 ;
            tree[nd] = 0 ;
         }
      }
      return ;
   }
   int mid = (l+r)/2 ;
   update(lc,l,mid,ql,qr,v);
   update(rc,mid+1,r,ql,qr,v);
   if(prop[nd] <= 0) {
      if(prop[lc] == -1 && prop[rc] == -1)
         prop[nd] = -1 ;

      else
         prop[nd] = max(prop[nd],0LL);
   }
   tree[nd] = (prop[nd] == -1 ? 0 : (prop[nd] == 0 ?
                                    tree[lc] + tree[rc] : r-l+1));
}
```

53. Rectangle Union Compress

```
vector<int> xpoints, ypoints ;
pair<int,int> tree[100000+7] ; // fs sum , sc prop
int is_seg[100000+7] ;

void relax(int nd,int l,int r,int v) {
   tree[nd].sc += v ;
   tree[nd].fs = 0 ;

   if(tree[nd].sc>0)
      tree[nd].fs = ypoints[r-1] - ypoints[l-1] ;

   else if(l != r) {
      tree[nd].fs = tree[2*nd].fs + tree[2*nd + 1].fs ;

      if(is_seg[nd]) {
         int mid = (l+r)/2 ;
         tree[nd].fs += ypoints[mid] - ypoints[mid-1] ;
      }
   }
}
void update(int nd,int l,int r,int i,int j,int v) {
```

```
    if(r<i || l>j)
        return ;

    if(l>=i && r<=j) {
        relax(nd,l,r,v);
        return ;
    }

    int mid = (l+r)/2, lc = 2*nd, rc = lc + 1 ;

    update(lc,l,mid,i,j,v);
    update(rc,mid+1,r,i,j,v);

    tree[nd].fs = tree[lc].fs + tree[rc].fs ;

    if(tree[nd].sc>0)
        tree[nd].fs = ypoints[r-1] - ypoints[l-1] ;

    if(i<=mid && j>=mid+1)
        is_seg[nd] += v ;

    if(is_seg[nd]>0 && tree[nd].sc == 0)
        tree[nd].fs += ypoints[mid] - ypoints[mid-1] ;
}
```

54. Li chao (Convex Hull Trick With Segment Tree)

```
// initially root is -1
// l = minimum possible value of x
// r = maximum possible value of x
// this is calculate for minimum
// for maximum change < sign
struct data {
    ll m, c ;
    int l, r ;
    data() {
        m = 0 ;
        c = 0 ;
        l = -1 ;
        r = -1 ;
    }
    data(ll m,ll c) {
        this->m = m ;
        this->c = c ;
        l = -1 ;
        r = -1 ;
    }
    ll cal_y(ll x) {
        return m*x + c ;
    }
};
data tree[MAXN] ;
bool vis[MAXN] ;
int id ;
int update(int nd,int l,int r,data line) {
    if(nd == -1) {
        tree[id] = line ;
        return id++;
    }

    if(tree[nd].cal_y(l)<=line.cal_y(l) && tree[nd].cal_y(r)<=line.cal_y(r))
        return nd ;
```

```
if(line.cal_y(l)<=tree[nd].cal_y(l) && line.cal_y(r) <=      tree[nd].cal_y(r)) {
      tree[id] = tree[nd] ;
      tree[id].c = line.c ;
      tree[id].m = line.m ;
      return id++;
   }

   int mid = (l+r)/2, lc = tree[nd].l, rc = tree[nd].r ;
   int nnd = id++ ;

   tree[nnd] = tree[nd] ;

   if(tree[nnd].cal_y(l)>line.cal_y(l))
      swap(tree[nnd].c,line.c), swap(tree[nnd].m,line.m) ;

   if(tree[nnd].cal_y(mid) <= line.cal_y(mid))        // Be careful about this
condition
      tree[nnd].r = update(tree[nnd].r,mid+1,r,line) ;

   else {
      swap(tree[nnd].c,line.c), swap(tree[nnd].m,line.m) ;
      tree[nnd].l = update(tree[nnd].l,l,mid,line);
   }

   return nnd ;
}

ll query(int nd,int l,int r,int x) {
   if(nd == -1)
      return inf ;

   int mid = (l+r)/2 ;

   if(mid>=x) // Be careful about this condition
      return min(tree[nd].cal_y(x),query(tree[nd].l,l,mid,x));

   return min(tree[nd].cal_y(x),query(tree[nd].r,mid+1,r,x));
}
```

55. MO's with Update

```
/**
Given an array. Two types of operation are supported .
0 A B -> summation of unique numbers which are divisible by 3
1 A B -> change value of index A to B
**/
const int sz = 200000+7 ;
map<int,int> mmap ;
int inp[sz], uind, qind, block_size, updates[sz], tarr[sz], freq[sz] ;
long long ans, qans[sz] ;
pair<int,int> updates_value[sz] ;
// first means previous , second means now
int rev[sz] ;
struct data {
   int l, r, tym, ind ;
   data() {}
   data(int l,int r,int tym,int ind) : l(l), r(r), tym(tym), ind(ind) {}
};
data query[sz] ;
bool cmp(data a,data b) {
   int b1 = a.l/block_size ;
   int b2 = b.l/block_size ;
   if(b1 == b2) {
```

```
        b1 = a.r/block_size ;
        b2 = b.r/block_size ;
        if(b1 == b2)
            return a.tym < b.tym ;
        return a.r < b.r ;
    }
    return a.l < b.l ;
}
void PUpdate(int ind,int l,int r) {
    int a ;
    if(updates[ind]>=l && updates[ind]<=r) {
        a = updates_value[ind].first ;
        freq[a]--;

        a = updates_value[ind].second ;
        freq[a]++;
    }
    inp[updates[ind]] = updates_value[ind].second ;
}
void UUpdate(int ind,int l,int r) {
    int a ;
    if(updates[ind]>=l && updates[ind]<=r) {
        a = updates_value[ind].second ;
        freq[a]--;

        a = updates_value[ind].first ;
        freq[a]++;
    }
    inp[updates[ind]] = updates_value[ind].first ;
}
void add(int ind) {
    int a = inp[ind] ;
    freq[a]++;
}

void rmv(int ind) {
    int a = inp[ind] ;
    freq[a]--;
}
void solve(int n,int q) {
    int typ, l, r ;
    qind = 0 ;
    uind = 0 ;
    for(int i=0 ; i<q ; i++) {
        scanf("%d",&typ);
        scanf("%d %d",&l,&r);
        if(typ == 0) { // updates query
            updates[uind] = l ;
            updates_value[uind] = make_pair(tarr[l],r) ;
            tarr[l] = r ;
            uind++;
        }
        else
            query[qind] = data(l,r,uind,qind), qind++;
    }
    block_size = cbrt(n);
    block_size = block_size*block_size ;
    sort(query,query+qind,cmp);
    int cur_time = 0 ;
    l = 1 ;
    r = 1 ;
```

```
for(int i=0 ; i<qind ; i++) {
   while(cur_time<query[i].tym) {
      PUpdate(cur_time,l,r-1);
      cur_time++;
   }

   while(cur_time>query[i].tym) {
      cur_time--;
      UUpdate(cur_time,l,r-1);
   }

   while(r<=query[i].r) {
      add(r);
      r++;
   }

   while(r>query[i].r+1) {
      r--;
      rmv(r);
   }

   while(l>query[i].l) {
      l--;
      add(l);
   }

   while(l<query[i].l) {
      rmv(l);
      l++;
   }
   qans[query[i].ind] = ans ;
   }
}
```

56. Kadane Algorithm of Maximum Sum (2-D)

```
// Maximum Sum Algo
int matrix[105][105], temp[110] ;
int finalleft, finalright, finaltop, finalbottom, n, start, finish ;
int main() {
   scanf("%d",&n);
   for(int i=0 ; i<n ; i++) {
      for(int j=0 ; j<n ; j++) {
         scanf("%d",&matrix[i][j]);
      }
   }
   find_maxsum();    return 0 ;
}
int kadane(int temp[]) {
   int sum = 0, maxsum = 0,local_start = 0 ;
   finish = -1 ;
   for(int i=0 ; i<n ; i++) {
      sum += temp[i];
      if(sum<0) {
         sum = 0 ;          local_start = i+1 ;
      }
      else if(sum>maxsum) {
         maxsum = sum ;          start = local_start ;
         finish = i ;
      }
   }
   if(finish != -1)       return maxsum;
   start = finish = 0 ;    sum = temp[0] ;
```

```
    for(int i=1 ; i<n ; i++) {
        if(temp[i]>sum) {
            sum = temp[i] ;          start = finish = i ;
        }
    }
    return sum ;
}
void find_maxsum() {
    int maxSum = 0, left, right, sum ;
    for(left = 0 ; left < n ; left++) {
        memset(temp,0,sizeof temp);
        for(right = left ; right<n ; right++) {
            for(int i=0 ; i<n ; i++)
                temp[i] += matrix[i][right];
            sum = kadane(temp);
            if(sum>maxSum) {
                maxSum = sum ;              finalleft = left ;
                finalright = right ;          finaltop = start ;
                finalbottom = finish ;
            }
        }
    }
    printf("%d\n",maxSum);
    printf("(Top, Left) (%d, %d)\n", finaltop, finalleft);
    printf("(Bottom, Right) (%d, %d)\n", finalbottom, finalright);
}
```

String Related Algorithm

57. Trie Tree

```
string str ;
int s_len, node[100000+5][53], id, cont[100000+5] ;
int new_node() {
   for(int i=0 ; i<52 ; i++)
      node[id][i] = 0 ;
   cont[id] = 0 ;
   return id++ ;
}
void add(int cur) {
   int a ;
   for(int i=0 ; i<s_len ; i++) {
      a = char_num(str[i]);
      if(node[cur][a] == 0)
         node[cur][a] = new_node();
      cur = node[cur][a] ;
   }
   cont[cur]++;
}
int query(int cur) {
   int a;
   for(int i=0 ; i<s_len ; i++) {
      a = char_num(str[i]);
      if(node[cur][a] == 0)
         return 0 ;
      cur = node[cur][a] ;
   }
   return cont[cur] ;
}
```

58. Trie XOR(Max/Min)

```
#define MAX 50001

struct trie {
   int cand[2];
   trie() {
      clrall(cand,-1);
   }
};

trie tree[MAX*32+7];
ll csum;
int tot_node;
void insert_trie(int root,ll val) {
   int i,j,k;
   int fbit;
   for(i = 31; i>=0; i--) {
      fbit=(int) ((val>>(ll) i)&1LL);
      if(tree[root].cand[fbit]==-1) {
         tree[root].cand[fbit] = ++tot_node;
      }
      root = tree[root].cand[fbit];
   }
   return ;
}

ll solve(int root,ll cval) {
   ll res=0;
   int fbit,cbit;
```

```
    int i,j,k;
    for(i = 31; i>=0; i--) {
        fbit=(int) ((cval>>(ll) i)&1LL);
        cbit=!(fbit);
        if(tree[root].cand[fbit]!=-1) {
            if(fbit) res|=(1LL << (ll) i);
            root=tree[root].cand[fbit];
        } else {
            if(cbit) res|=(1LL << (ll) i);
            root=tree[root].cand[cbit];
        }
    }
    return res;
}
ll max_val(ll val) {
    int i,j,k;
    ll ret=0;
    int gbit;
    for(i = 31; i>=0; i--) {
        gbit=(int) ((val>>(ll) i)&1LL);
        if(!gbit) ret|=(1LL << (ll) i);
    }
    return ret;
}
ll min_val(ll val) {
    int i,j,k;
    ll ret=0;
    int gbit;
    for(i = 31; i>=0; i--) {
        gbit=(int) ((val>>(ll) i)&1LL);
        if(gbit) ret|=(1LL << (ll) i);
    }
    return ret;
}
int main() {
    ll val;
    int test,cas=0,i,j,k,n;
    ll ansmx,ansmn;
    cin>>test;
    rep(i,0,MAX*32+7) tree[i]=trie();
    while(test--) {
        cin>>n;
        tot_node=0;
        csum=0LL;
        ansmx=0LL;
        ansmn=(1LL<<50LL);
        insert_trie(0,csum);
        rep(i,1,n+1) {
            cin>>val;
            csum=csum xor val;
            val=max_val(csum);
            ansmx=max(ansmx,solve(0,val) xor csum);
            val=min_val(csum);
            ansmn=min(ansmn,solve(0,val) xor csum);
            insert_trie(0,csum);
        }
        cas++;
        cout<<"Case "<<cas<<": "<<ansmx<<" "<<ansmn<<endl;
        rep(i,0,tot_node+4) tree[i]=trie();
    }
    return 0;
}
```

59. Trie Tree (Jubair)

```
string str ;
int s_len , node[100000+5][53] , id , cont[100000+5] ;

int new_node()
{
   for(int i=0 ; i<52 ; i++)
      node[id][i] = 0 ;

   cont[id] = 0 ;

   return id++ ;
}

void add(int cur)
{
   int a ;
   for(int i=0 ; i<s_len ; i++)
   {
      a = char_num(str[i]);

      if(node[cur][a] == 0)
         node[cur][a] = new_node();

      cur = node[cur][a] ;
   }

   cont[cur]++;
}

int query(int cur)
{
   int a;
   for(int i=0 ; i<s_len ; i++)
   {
      a = char_num(str[i]);

      if(node[cur][a] == 0)
         return 0 ;

      cur = node[cur][a] ;
   }
   return cont[cur] ;
}
```

60. Persistent Trie (NEW)

```
#include <bits/stdc++.h>
using namespace std;
/*Type 0: Add the integer number x at the end of the array.
Type 1: On the interval L..R find a number y, to maximize (x xor y).
Type 2: Delete last k numbers in the array
Type 3: On the interval L..R, count the number of integers less than or
equal to x.
Type 4: On the interval L..R, find the kth smallest integer (kth order
statistic).*/
using namespace std;
#define pb push_back
#define mx 500000*15
vector<int>vc;
int
id,tr[mx][2],ara[mx],root[600000],num,tree[mx],lefto[mx],righto[mx],root
2[600000],ck,ans,powll[30];
int build(int st,int en)
{
   int pse=++num;
```

```
    if(st==en)
        return pse;
    int mid=(st+en)/2;
    lefto[pse]=build(st,mid);
    righto[pse]=build(mid+1,en);
    return pse;
}
int update(int node,int st,int en,int l)
{
    if(st>l or en<l)
        return node;
    int pse=++num;
    if(st==en)
    {
        tree[pse]=tree[node]+1;
        return pse;
    }
    int mid=(st+en)/2;
    lefto[pse]=update(lefto[node],st,mid,l);
    righto[pse]=update(righto[node],mid+1,en,l);
    tree[pse]=tree[lefto[pse]]+tree[righto[pse]];
    return pse;
}
int query(int node,int node2,int st,int en,int l,int r,int need)
{
    if(st>r or en<l)
        return 0;
    if((tree[node2]-tree[node])<need&&st>=l and en<=r)
        return tree[node2]-tree[node];
    if(st==en)
    {
        ck=st;
        return 0;
    }
    int mid=(st+en)/2;
    int p=query(lefto[node],lefto[node2],st,mid,l,r,need);
    if(ck!=-1)
        return 0;
    query(righto[node],righto[node2],mid+1,en,l,r,need-p);
}
int query2(int node,int node2,int st,int en,int l,int r)
{
    if(st>r or en<l)
        return 0;
    if(st>=l and en<=r)
    {
        return tree[node2]-tree[node];
    }
    int mid=(st+en)/2;
    int p=query2(lefto[node],lefto[node2],st,mid,l,r);
    int q=query2(righto[node],righto[node2],mid+1,en,l,r);
    return p+q;
}
int rec(int cur,int pos)
{
    int pse=++id;
    for(int i=0; i<=1; i++)
    {
        // ara[pse][i]=ara[cur][i];
        tr[pse][i]=tr[cur][i];
    }
    ara[pse]=ara[cur]+1;
    if(pos==20)
    {
        return pse;
    }
```

```
   // ara[pse][vc[pos]]+=1;
   tr[pse][vc[pos]]=rec(tr[cur][vc[pos]],pos+1);
   return pse;
}
void query(int curl,int curr,int pos)
{
   if(pos==20)
      return ;
   if(vc[pos]==0)
   {
      if(ara[tr[curr][1]]-ara[tr[curl][1]])
      {
         ans+=powll[19-pos];
         query(tr[curl][1],tr[curr][1],pos+1);
         return ;
      }
      query(tr[curl][0],tr[curr][0],pos+1);
      return ;
   }
   if(ara[tr[curr][0]]-ara[tr[curl][0]])
   {
      //   ans+=powll[19-pos];
      query(tr[curl][0],tr[curr][0],pos+1);
      return;
   }
   ans+=powll[19-pos];
   query(tr[curl][1],tr[curr][1],pos+1);
   return ;
}
int n=500000,st;
void feel(int x)
{
   vc.clear();
   while(x!=0)
   {
      vc.pb(x%2);
      x=x/2;
   }
   while(vc.size()<20)
      vc.pb(0);
   reverse(vc.begin(),vc.end());
}
int handle0()
{
   int x;
   scanf("%d",&x);
   st++;
   root2[st]=update(root2[st-1],1,n,x);
   feel(x);
   root[st]=rec(root[st-1],0);
}
int handle1()
{
   int x,l,r;
   scanf("%d%d%d",&l,&r,&x);
   feel(x);
   ans=0;
   query(root[l-1],root[r],0);
   printf("%d\n",ans);
}
int handle2()
{
   int k;
   scanf("%d",&k);
   st-=k;
}
```

```
int handle4()
{
   int l,r,k;
   scanf("%d%d%d",&l,&r,&k);
   ck=-1;
   //  cout<<tree[root2[r]]<<endl;
   query(root2[l-1],root2[r],1,n,1,n,k);
   printf("%d\n",ck);
}
int handle3()
{
   int l,r,k;
   scanf("%d%d%d",&l,&r,&k);
   l=query2(root2[l-1],root2[r],1,n,1,k);
   printf("%d\n",l);
}
int main()
{
   int i,j,k,l,m,ty;
   powll[0]=1;
   for(int i=1; i<=20; i++)
   {
      powll[i]=powll[i-1]*2;
   }
   root2[0]=build(1,n);
   scanf("%d",&m);
   for(i=1; i<=m; i++)
   {
      scanf("%d",&ty);
      if(ty==0)
         handle0();
      if(ty==1)
         handle1();
      if(ty==2)
         handle2();
      if(ty==3)
         handle3();
      if(ty==4)
         handle4();
   }
}
```

61. Persistent Trie (OLD)

```
#define pb push_back
#define mx 500000*15
vector<int>vc;
int
id,tr[mx][2],ara[mx][2],root[600000],num,tree[mx],lefto[mx],righto[mx],r
oot2[600000],ck,ans,powll[30];
int build(int st,int en)
{
   int pse=++num;
   if(st==en)
      return pse;
   int mid=(st+en)/2;
   lefto[pse]=build(st,mid);
   righto[pse]=build(mid+1,en);
   return pse;
}
int update(int node,int st,int en,int l)
{
   if(st>l or en<l)
      return node;
   int pse=++num;
   if(st==en)
   {
```

```
        tree[pse]=tree[node]+1;
        return pse;
    }
    int mid=(st+en)/2;
    lefto[pse]=update(lefto[node],st,mid,l);
    righto[pse]=update(righto[node],mid+1,en,l);
    tree[pse]=tree[lefto[pse]]+tree[righto[pse]];
    return pse;
}
int query(int node,int node2,int st,int en,int l,int r,int need)
{
    if(st>r or en<l)
        return 0;
    if((tree[node2]-tree[node])<need&&st>=l and en<=r)
        return tree[node2]-tree[node];
    if(st==en)
    {
        ck=st;
        return 0;
    }
    int mid=(st+en)/2;
    int p=query(lefto[node],lefto[node2],st,mid,l,r,need);
    if(ck!=-1)
        return 0;
    query(righto[node],righto[node2],mid+1,en,l,r,need-p);
}
int query2(int node,int node2,int st,int en,int l,int r)
{
    if(st>r or en<l)
        return 0;
    if(st>=l and en<=r)
    {
        return tree[node2]-tree[node];
    }
    int mid=(st+en)/2;
    int p=query2(lefto[node],lefto[node2],st,mid,l,r);
    int q=query2(righto[node],righto[node2],mid+1,en,l,r);
    return p+q;
}
int rec(int cur,int pos)
{
    int pse=++id;
    for(int i=0; i<=1; i++)
    {
        ara[pse][i]=ara[cur][i];
        tr[pse][i]=tr[cur][i];
    }
    if(pos==20)
    {
        return pse;
    }
    ara[pse][vc[pos]]+=1;
    tr[pse][vc[pos]]=rec(tr[cur][vc[pos]],pos+1);
    return pse;
}
void query(int curl,int curr,int pos)
{
    if(pos==20)
        return ;
    if(vc[pos]==0)
    {
        if(ara[curr][1]-ara[curl][1])
        {
            ans+=powll[19-pos];
            query(tr[curl][1],tr[curr][1],pos+1);
            return ;
```

```
        }
        query(tr[curl][0],tr[curr][0],pos+1);
        return ;
    }
    if(ara[curr][0]-ara[curl][0])
    {
        //  ans+=powll[19-pos];
        query(tr[curl][0],tr[curr][0],pos+1);
        return;
    }
    ans+=powll[19-pos];
    query(tr[curl][1],tr[curr][1],pos+1);
    return ;
}
int n=500000,st;
void feel(int x)
{
    vc.clear();
    while(x!=0)
    {
        vc.pb(x%2);
        x=x/2;
    }
    while(vc.size()<20)
        vc.pb(0);
    reverse(vc.begin(),vc.end());
}
int handle0()
{
    int x;
    scanf("%d",&x);
    st++;
    root2[st]=update(root2[st-1],1,n,x);
    feel(x);
    root[st]=rec(root[st-1],0);
}
int handle1()
{
    int x,l,r;
    scanf("%d%d%d",&l,&r,&x);
    feel(x);
    ans=0;
    query(root[l-1],root[r],0);
    printf("%d\n",ans);
}
int handle2()
{
    int k;
    scanf("%d",&k);
    st-=k;
}
int handle4()
{
    int l,r,k;
    scanf("%d%d%d",&l,&r,&k);
    ck=-1;
    //  cout<<tree[root2[r]]<<endl;
    query(root2[l-1],root2[r],1,n,1,n,k);
    printf("%d\n",ck);
}
int handle3()
{
    int l,r,k;
    scanf("%d%d%d",&l,&r,&k);
    l=query2(root2[l-1],root2[r],1,n,1,k);
    printf("%d\n",l);
```

```
}
int main()
{
   int i,j,k,l,m,ty;
   powll[0]=1;
   for(int i=1; i<=20; i++)
   {
      powll[i]=powll[i-1]*2;
   }
   root2[0]=build(1,n);
   scanf("%d",&m);
   for(i=1; i<=m; i++)
   {
      scanf("%d",&ty);
      if(ty==0)
         handle0();
      if(ty==1)
         handle1();
      if(ty==2)
         handle2();
      if(ty==3)
         handle3();
      if(ty==4)
         handle4();
   }
}
```

62. Suffix Array (TeamX)

```
#define MAX 100000

string text;
int revSA[MAX],SA[MAX];
int cnt[MAX], nxt[MAX];
bool bh[MAX],b2h[MAX];
int LCP[MAX];

bool cmp(const int &i,const int &j) {
   return text[i]<text[j];
}
void sortFirstChar(int n) {
   /// sort for the first char  ...
   for(int i =0 ; i<n ; i++)
      SA[i] = i;
   sort(SA,SA+n,cmp);

   ///indentify the bucket ........
   for(int i=0 ; i<n ; i++) {
      bh[i] = (i==0  || text[SA[i]]!=text[SA[i-1]]);
      b2h[i] = false;
   }
   return;
}
int CountBucket(int n) {
   int bucket = 0;
   for(int i =0,j; i<n ; i=j) {
      j = i+1;
      while(j<n && bh[j]==false) j++;
      nxt[i] = j;
      bucket++;
   }
   return bucket;
}
void SetRank(int n) {
   for(int i = 0 ; i<n ; i=nxt[i]) {
```

```
        cnt[i] = 0;
        for(int j =i ; j<nxt[i] ; j++) {
            revSA[SA[j]] = i;
        }
    }
    return;
}
void findNewRank(int l,int r,int step) {
    for(int j = l  ; j<r ; j++) {
        int pre = SA[j] - step;
        if(pre>=0) {
            int head = revSA[pre];
            revSA[pre] = head+cnt[head]++;
            b2h[revSA[pre]] = true;
        }
    }
    return;
}
void findNewBucket(int l,int r,int step) {
    for(int j = l  ; j<r ; j++) {
        int pre = SA[j] - step;
        if(pre>=0 && b2h[revSA[pre]]) {
            for(int k = revSA[pre]+1 ; b2h[k] && !bh[k] ; k++) b2h[k] = false;
        }
    }
    return;
}
void buildSA(int n) {
    ///start sorting in logn step ...
    sortFirstChar(n);
    for(int h =1 ; h<n ; h<<=1) {
        if(CountBucket(n)==n) break;
        SetRank(n);
        /// cause n-h suffix must be sorted
        b2h[revSA[n-h]] = true;
        cnt[revSA[n-h]]++;
        for(int i = 0 ; i<n ; i=nxt[i]) {
            findNewRank(i,nxt[i], h);
            findNewBucket(i, nxt[i], h);
        }
        ///set the new sorted suffix array ...
        for(int i =0 ; i<n ; i++) {
            SA[revSA[i]] = i;
            bh[i] |= b2h[i]; ///new bucket ....
        }
    }
    return;
}

void buildLCP(int n) {
    int len = 0;
    for(int i = 0 ; i<n ; i++)
        revSA[SA[i]] = i;
    for(int i =0 ; i< n ; i++) {
        int k = revSA[i];
        if(k==0) {
            LCP[k] = 0;
            continue;
        }
        int j = SA[k-1];
        while(text[i+len]==text[j+len]) len++;
        LCP[k] = len;
```

```
      if(len) len--;
   }
   return;
}

void printSA(int n) {
   for(int i=0; i<n; i++) printf("%2d ",SA[i]),cout<<text.substr(SA[i])<<endl;
   puts("");
   for(int i=1; i<n; i++) printf("%2d\n",LCP[i]);
   puts("");
   return ;
}
int main() {
   string a,b;
   int n,p,q;
   int tcase,cas=1;
   scanf("%d",&tcase);
   while(tcase--) {
//       cin>>a>>b;
//       text=a+"$"+b;
       cin>>text;
       buildSA(SZ(text));
       buildLCP(SZ(text));
       printSA(SZ(text));
       int r=0;
       int n=SZ(text);
       for(int i=0; i<n; i++) {
          r+=(n-i);
          r-=LCP[i];
       }
       deb(r);
   }
   return 0;
}
```

63. Suffix Array (Jubair)

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "algorithm"
using namespace std;
const int MAX = 100010;
char txt[MAX]; //input
int iSA[MAX], SA[MAX]; //output
int cnt[MAX], next[MAX]; //internal
bool bh[MAX], b2h[MAX];

// Compares two suffixes according to their first characters
bool smaller_first_char(int a, int b)
{
   return txt[a] < txt[b];
}

void suffixSort(int n)
{
   //sort suffixes according to their first characters
   for (int i=0; i<n; ++i)
   {
      SA[i] = i;
   }
   sort(SA, SA + n, smaller_first_char);
   //{SA contains the list of suffixes sorted by their first character}
```

```
   for (int i=0; i<n; ++i)
   {
      bh[i] = i == 0 || txt[SA[i]] != txt[SA[i-1]];
      b2h[i] = false;
   }

   for (int h = 1; h < n; h <<= 1)
   {
      //{bh[i] == false if the first h characters of SA[i-1] == the first h
characters of SA[i]}
      int buckets = 0;
      for (int i=0, j; i < n; i = j)
      {
         j = i + 1;
         while (j < n && !bh[j])
            j++;
         next[i] = j;
         buckets++;
      }
      if (buckets == n)
         break; // We are done! Lucky bastards!
      //{suffixes are separted in buckets containing txtings starting with
the same h characters}

      for (int i = 0; i < n; i = next[i])
      {
         cnt[i] = 0;
         for (int j = i; j < next[i]; ++j)
         {
            iSA[SA[j]] = i;
         }
      }

      cnt[iSA[n - h]]++;
      b2h[iSA[n - h]] = true;
      for (int i = 0; i < n; i = next[i])
      {
         for (int j = i; j < next[i]; ++j)
         {
            int s = SA[j] - h;
            if (s >= 0)
            {
               int head = iSA[s];
               iSA[s] = head + cnt[head]++;
               b2h[iSA[s]] = true;
            }
         }
         for (int j = i; j < next[i]; ++j)
         {
            int s = SA[j] - h;
            if (s >= 0 && b2h[iSA[s]])
            {
               for (int k = iSA[s]+1; !bh[k] && b2h[k]; k++)
                  b2h[k] = false;
            }
         }
      }
      for (int i=0; i<n; ++i)
      {
         SA[iSA[i]] = i;
         bh[i] |= b2h[i];
```

```
      }
   }
   for (int i=0; i<n; ++i)
   {
      iSA[SA[i]] = i;
   }
}
int lcp[MAX];
void getlcp(int n)
{
   for (int i=0; i<n; ++i)
      iSA[SA[i]] = i;

   lcp[0] = 0;

   for (int i=0, h=0; i<n; ++i)
   {
      if (iSA[i] > 0)
      {
         int j = SA[iSA[i]-1];
         while (i + h < n && j + h < n && txt[i+h] == txt[j+h])
            h++;
         lcp[iSA[i]] = h;

         if (h > 0)
            h--;
      }
   }
}
// End of longest common prefixes algorithm
int main()
{
   int len;
   gets(txt);
   len = strlen(txt);
   suffixSort(len);
   for (int i = 0; i < len; ++i)
   {
      printf("%d\n",SA[i] );
   }
   return 0;
}
```

64. Suffix Array (DC3)

```
#include<bits/stdc++.h>
#define N 2000005
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
using namespace std;
int wa[N],wb[N],wv[N],wS[N];
int rnk[N],height[N];   // rank hocche inverse sa, height hocche lcp array
int sa[N],r[N];
char c[N];

int c0(int *y,int a,int b)
{
   return y[a]==y[b]&&y[a+1]==y[b+1]&&y[a+2]==y[b+2];
}
int c12(int k,int *y,int a,int b)
{
   if(k==2)
      return y[a]<y[b]||y[a]==y[b]&&c12(1,y,a+1,b+1);
```

```
    else
        return y[a]<y[b]||y[a]==y[b]&&wv[a+1]<wv[b+1];
}
void sort(int *r,int *a,int *b,int n,int m)
{
    int i;
    for(i=0; i<n; i++)
        wv[i]=r[a[i]];
    for(i=0; i<m; i++)
        wS[i]=0;
    for(i=0; i<n; i++)
        wS[wv[i]]++;
    for(i=1; i<m; i++)
        wS[i]+=wS[i-1];
    for(i=n-1; i>=0; i--)
        b[--wS[wv[i]]]=a[i];
    return;
}
void build_suffix(int *r,int *sa,int n,int m)
{
    int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
    r[n]=r[n+1]=0;
    for(i=0; i<n; i++)
        if(i%3!=0)
            wa[tbc++]=i;
    sort(r+2,wa,wb,tbc,m);
    sort(r+1,wb,wa,tbc,m);
    sort(r,wa,wb,tbc,m);
    for(p=1,rn[F(wb[0])]=0,i=1; i<tbc; i++)
        rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
    if(p<tbc)
        build_suffix(rn,san,tbc,p);
    else
        for(i=0; i<tbc; i++)
            san[rn[i]]=i;
    for(i=0; i<tbc; i++)
        if(san[i]<tb)
            wb[ta++]=san[i]*3;
    if(n%3==1)
        wb[ta++]=n-1;
    sort(r,wb,wa,ta,m);
    for(i=0; i<tbc; i++)
        wv[wb[i]=G(san[i])]=i;
    for(i=0,j=0,p=0; i<ta && j<tbc; p++)
        sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
    for(; i<ta; p++)
        sa[p]=wa[i++];
    for(; j<tbc; p++)
        sa[p]=wb[j++];
    return;
}
void get_lcp(int n)
{
    int i,j,k=0;
    for(i=0; i<=n; i++)
        rnk[sa[i]]=i;
    for(i=0; i<n; height[rnk[i++]]=k)
        for(k?k--:0,j=sa[rnk[i]-1]; r[i+k]==r[j+k]; k++);
    return;
}

int main()
```

```
{
   long long mot;
   int t;
   scanf("%d",&t);
   getchar();
   while(t--)
   {
      scanf("%s", c);
      int n = strlen(c);
      mot=0;
      for(int i = 0; i<n; i++)
         r[i] = c[i],mot+=(1ll*(i+1));
      r[n] = 0;
      build_suffix(r, sa, n+1, 256);
      get_lcp(n);
      for(int i=0; i<=n; i++)
         mot-=(1ll*height[i]);
      printf("%lld\n",mot);
   }
   return 0;
}
```

65. Suffix Automata

```
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
#define mp make_pair
struct node
{
   int len,prio,Q;
   node() {}
   node(int _len,int _prio,int _Q)
   {
      len=_len;
      prio=_prio;
      Q=_Q;
   }
   bool operator<(const node &P)const
   {
      if(len==P.len)
         return prio<P.prio;
      return len<P.len;
   }
};
vector<node>occur[300000];
vector<pair<int,int>>my;
struct state
{
   int len, link;
   map<char,int>next;
};

const int MAXLEN = 500500;
state st[MAXLEN];

int out[MAXLEN],vis[MAXLEN],cnt[MAXLEN];
int sz, last;

void sa_init()
{
   sz = last = 0;
   st[0].len = 0;
   st[0].link = -1;
   ++sz;
}
```

```
void sa_extend (char c)
{
   int cur = sz++;
   cnt[cur]=1;

   st[cur].len = st[last].len + 1;
   int p;
   for (p=last; p!=-1 && !st[p].next.count(c); p=st[p].link)
      st[p].next[c] = cur;
   if (p == -1)
      st[cur].link = 0;
   else
   {
      int q = st[p].next[c];
      if (st[p].len + 1 == st[q].len)
         st[cur].link = q;
      else
      {
         int clone = sz++;
         st[clone].len = st[p].len + 1;
         st[clone].next = st[q].next;
         st[clone].link = st[q].link;
         for (; p!=-1 && st[p].next[c]==q; p=st[p].link)
            st[p].next[c] = clone;
         st[q].link = st[cur].link = clone;
      }
   }
   last = cur;
}
char str[300000];
int main()
{
   int i,j,k,l,m,n;
   scanf("%s",str);
   m=strlen(str);
   sa_init();
   for(int i=0; str[i]; i++)
   {
      sa_extend(str[i]);
   }
   for(int i=1; i<sz; i++)
   {
      my.pb(mp(st[i].len,i));
   }
   sort(my.begin(),my.end());
   for(int i=my.size()-1; i>=0; i--)
   {
      l=my[i].second;
      cnt[st[l].link]+=cnt[l];
      int lo=st[st[l].link].len+1;
      int high=st[l].len;
      occur[cnt[l]].pb(node(lo,1,-1));
      occur[cnt[l]].pb(node(high,3,-1));
   }
   scanf("%d",&n);
   for(int i=1; i<=n; i++)
   {
      scanf("%d%d",&l,&k);
      vis[k]=1;
      occur[k].pb(node(l,2,i));
   }
   for(int i=1; i<=m; i++)
   {
      if(vis[i]==0)
         continue;
```

```
    int ans=0;
    sort(occur[i].begin(),occur[i].end());
    for(int j=0; j<occur[i].size(); j++)
    {
      if(occur[i][j].prio==3)
      {
        ans--;
        continue;
      }
      if(occur[i][j].prio==1)
      {
        ans++;
        continue;
      }
      out[occur[i][j].Q]=ans;
    }
  }
  for(int i=1; i<=n; i++)
  {
    printf("%d\n",out[i]);
  }
}
```

66. Suffix Tree (TeamX)

```
int num[sz], match, node, n_d, graph[30*sz][30], link[30*sz] ;
jora_int edge[30*sz] ;
int add_edge(int st,int ln) {
  edge[n_d].fs = st ;
  edge[n_d].sc = ln ;
  for(int i=0 ; i<30 ; i++)
    graph[n_d][i] = 0 ;
  return n_d++;
}
void _jump(int pos) {
  while(match > edge[graph[node][num[pos+1-match]]].sc) {
    node = graph[node][num[pos+1-match]] ;
    match -= edge[node].sc ;
  }
}
void add_char(int pos) {
  int last = 0, a = num[pos], cur_ed, m_ed, u ;
  match++;
  while(match>0) {
    _jump(pos);
    cur_ed = num[pos - match + 1] ;
    int& v = graph[node][cur_ed] ;
    m_ed = num[edge[v].fs + match - 1] ;
    if(v == 0) {
      v = add_edge(pos-match+1,Max);//deb(pos,v);
      link[last] = node ;
      last = 0 ;
    } else if(a == m_ed) {
      link[last] = node ;
      return ;
    } else {
      u = add_edge(edge[v].fs,match-1);
      graph[u][a] = add_edge(pos,Max);
      graph[u][m_ed] = v ;
//if(edge[v].sc<match-1) deb(v,edge[v].sc,pos,match),wait;
      edge[v].fs += match - 1;
      edge[v].sc -= match - 1;
      v = u ;
      link[last] = u ;
      last = u ;
```

```
        }
        if(node == 0)
            match--;
        else
            node = link[node] ;
    }
}
void reset() {
    node = 0 ;
    n_d = 0 ;
    match = 0 ;
    add_edge(0,Max);
}
void print(int nd) {
    if(nd)
        pf("%d %d %d\n",nd,edge[nd].fs,edge[nd].sc);
    for(int i=0 ; i<29 ; i++) {
        if(graph[nd][i] > 0) {
            deb(nd,i);
            print(graph[nd][i]);
        }
    }
}
int main() {
    string str ;
    cin>>str ;
    reset();
    for(int i=0 ; i<str.length() ; i++) {
        num[i] = str[i] - 'a' ;
        add_char(i);
    }
    print(0);
    return 0 ;
}
```

67. KMP

```
vector<int> prefix_cal(char str[]) {
    int l = strlen(str+1) ;
    vector<int>prefix(l+1) ;
    prefix[1] = 0 ;
    int k = 0 ;
    for(int i=2 ; i<=l ; i++) {
        while(k>0 and str[i] != str[k+1])
            k = prefix[k];
        if(str[k+1] == str[i])
            k++;
        prefix[i] = k ;
    }
    return prefix;
}
vector<int> match_prefix(char par[],char str[]) {
    int l1 = strlen(str+1), l2 = strlen(par+1), k = 0 ;
    vector<int>prefix, match ;
    prefix = prefix_cal(par) ;
    for(int i = 1 ; i<=l1 ; i++) {
        while(k>0 and str[i] != par[k+1])
            k = prefix[k];
        if(str[i] == par[k+1])
            k++;
        if(k == l2) {
            match.pb(i-k);
            k = prefix[k] ;
```

```
      }
   }
   return match ;
}
```

68. Minimum Expression and ExKmp

```
int nxt[N],ex_a[N],exb[N];
void getnext(char *s) {
   int len = strlen(s),cur = 0;
   nxt[0] = len;
   while(cur < len&&s[cur]==s[cur+1])cur++;
   nxt[1] = cur;
   cur = 1;
   for(int k = 2; k<len; k++) {
      int p = cur + nxt[cur] - 1,L = nxt[k-cur];
      if(k + L -1 >= p) {
         int j = (p-k+1)>0?(p-k+1):0;
         while(k+j<len&&s[k+j]==s[j])j++;
         nxt[k] = j;
         cur = k;
      } else
         nxt[k] = L;
   }
}
/*  exkmp return match for each position between strings
suffix starts from this position and pattern  */
void exkmp(char *s1,char *s2,int *ex) { // s1 is main string, s2 pattern
   getnext(s2);
   int l1 = strlen(s1),l2 = strlen(s2),cur = 0;
   while(cur < min(l1,l2)&&s1[cur]==s2[cur])cur++;
   ex[0] = cur;
   cur = 0;
   for(int k = 1; k < l1; k++) {
      int p = cur + ex[cur] - 1,L = nxt[k-cur];
      if(k + L -1 >= p) {
         int j = (p-k+1)>0?(p-k+1):0;
         while(k+j<l1&&j<l2&&s1[k+j]==s2[j])j++;
         ex[k] = j;
         cur = k;
      } else
         ex[k] = L;
   }
}
int MinRep(char *s) { // return position from where this cyclic string is
Lexicographical minimum
   int i = 0,j = 1,k = 0,t,len = strlen(s);
   while(i<len&&j<len&&k<len) {
      t = s[(i+k)%len] - s[(j+k)%len];
      if(t==0)k++;
      else {
         if(t>0)
            i += k + 1;
         else
            j += k + 1;
         if(i==j)j++;
         k = 0;
      }
   }
   return min(i,j);
}
```

69. Aho Chorasic

```
#define pb push_back
char text[2000000],str[505][505];
vector<int>vc[400000];
int
nwnode[400000][27],backnode[400000],cnt[400000],vis[400000],id=0;

int newnode()
{
   id++;
   for(int i=1; i<=26; i++)
   {
      nwnode[id][i]=0;
   }
   vis[id]=0;
   cnt[id]=0;
   vc[id].clear();
   return id;
}
void build(int n)
{
   int root=newnode(),p;
   queue<int>q;
   for(int i=1; i<=n; i++)
   {
      p=root;
      for(int j=0; str[i][j]; j++)
      {
         int c=str[i][j]-96;
         // cout<<p<<" "<<c<<endl;
         if(!nwnode[p][c])
            nwnode[p][c]=newnode();
         p=nwnode[p][c];
         //cout<<p<<endl;
      }
   }
   for(int i=1; i<=26; i++)
   {
      if(!nwnode[root][i])
         nwnode[root][i]=root;
      else
      {
         q.push(nwnode[root][i]);
         backnode[nwnode[root][i]]=1;
      }
   }
   int u,v,w,c;
   while(!q.empty())
   {
      u=q.front();
      // cout<<u<<"sss"<<endl;
      q.pop();
      for(int i=1; i<=26; i++)
      {
         if(!nwnode[u][i])
            continue;
         w=backnode[u];
         v=nwnode[u][i];
         // cout<<v<<" "<<u<<" "<<w<<endl;
         while(nwnode[w][i]==0)
         {
```

```
            w=backnode[w];
        }
        int c=nwnode[w][i];
        backnode[v]=w=c;
        vc[w].pb(v);
        q.push(v);
      }
   }
}
void ahokoracik()
{
   int p=1,c;
   for(int i=0; text[i]; i++)
   {
      c=text[i]-96;
      while(!nwnode[p][c])
         p=backnode[p];
      p=nwnode[p][c];
      cnt[p]++;
   }
}
int dfs(int p)
{
   if(vis[p]==1)
      return cnt[p];
   for(int i=0; i<vc[p].size(); i++)
   {
      int w=vc[p][i];
      cnt[p]+=dfs(w);
   }
   vis[p]=1;
   return cnt[p];
}
int main()
{
   int i,j,k,l,m,n,p,c,test,casio=1;
   scanf("%d",&test);
   while(test--)
   {
      scanf("%d",&n);
      id=0;
      scanf("%s",text);
      for(int i=1; i<=n; i++)
      {
         scanf("%s",str[i]);
      }
      build(n);
      ahokoracik();
      printf("Case %d:\n",casio++);
      for(int i=1; i<=n; i++)
      {
         p=1;
         for(int j=0; str[i][j]; j++)
         {
            c=str[i][j]-96;
            p=nwnode[p][c];
         }
         //  cout<<p<<endl;
         printf("%d\n",dfs(p));
      }
   }
}
```

70. Dynamic Aho Chorasic

```
const int INF = 2000000009;
const int MX = 100005;
const double EPS = 1e-9;
const int MOD = 1000000007;

/**********************Code starts here*********************************/

struct Ahocorasik
{
	struct Node
	{
		unordered_map<int,int> next;
		int cnt,sufflink,isleaf;
		Node(){
			cnt = sufflink = isleaf = 0;
		}
	};
	vector<Node> trie;
	int size,wordcnt;
	Ahocorasik(){
		wordcnt = 0;
		size = 0;
		newNode();
	}
	void reset(){
		trie.clear();
		wordcnt = 0;
		size = 0;
		newNode();
	}
	int newNode(){
		trie.push_back(Node());
		return size++;
	}
	int insert(char *s)
	{
		int cur = 0;
		for (int i = 0;s[i];i++){
			int let = s[i] - 'a';
			if (!trie[cur].next.count(let)) trie[cur].next[let] = newNode();
			cur = trie[cur].next[let];
		}
		trie[cur].isleaf += 1;
		return cur;
	}
	void buildFail(){
		queue<int> q;
		for (int i = 0;i<26;i++) if (trie[0].next.count(i)){
			q.push(trie[0].next[i]);
			trie[trie[0].next[i]].cnt = trie[trie[0].next[i]].isleaf;
		}
		while (!q.empty()){
			int u = q.front(); q.pop();
			for (int i = 0;i<26;i++) if (trie[u].next.count(i)){
				int v = trie[u].next[i];
				int f = trie[u].sufflink;
				while (f && trie[f].next.count(i)==0) f = trie[f].sufflink;
```

```
                        if (trie[f].next.count(i)) f = trie[f].next[i];
                        trie[v].sufflink = f;
                        trie[v].cnt= trie[f].cnt +trie[v].isleaf;
                        q.push(v);
                    }
                }
        }
        long long find(char *s){
                int cur = 0; long long ret = 0;
                for (int i = 0;s[i];i++){
                        int let = s[i] - 'a';
                        while(cur && trie[cur].next.count(let)==0 ) cur =
trie[cur].sufflink;
                        if (trie[cur].next.count(let)) cur =
trie[cur].next[let];
                        ret+=trie[cur].cnt;
                }
                return ret;
        }
};
struct OnlineAhocorasik
{
        Ahocorasik aho[23];
        OnlineAhocorasik(){}
        void insert(char *s)
        {
                int i = 0;
                int cnt = 1;
                for (i = 0;i<23;i++){
                        cnt+=aho[i].wordcnt;
                        if ((1<<i)>=cnt) {
                                break;
                        }
                }
                aho[i].insert(s);
                for (int j = 0;j<i;j++){
                        merge(aho[i],aho[j]);
                        aho[j].reset();
                }
                aho[i].wordcnt = cnt;
                aho[i].buildFail();
        }
        void merge(Ahocorasik &a,Ahocorasik &b,int cur1 = 0,int cur2 =
0){
                a.trie[cur1].isleaf += b.trie[cur2].isleaf;
                for (auto  i : b.trie[cur2].next){
                        if (a.trie[cur1].next.count(i.FF)==0)
                                a.trie[cur1].next[i.FF] = a.newNode();
                        merge(a,b,a.trie[cur1].next[i.FF],i.SS);
                }
        }
        long long find(char *s){
                long long  ret = 0;
                for (int i = 0;i<23;i++){
                        ret+=aho[i].find(s);
                }
                return ret;
        }
};
char s[3*MX];
int main()
{
```

```
	//std::ios_base::sync_with_stdio(false);
	OnlineAhocorasik a,b;
	int m = II();
	while (m--)
	{
		int t = II();
		scanf("%s",s);
		if (t==1) a.insert(s);
		else if (t==2) b.insert(s);
		else {
			printf("%lld\n",a.find(s)-b.find(s) );
			cout.flush();
		}
	}
	return 0;
}
```

71. Manachers

```
char str[sz], fstr[2*sz] ;
int len, p[2*sz] ;
void reset() {
   len = strlen(str);
   fstr[0] = '^' ;
   p[0] = 0 ;
   int k = 1 ;
   for(int i=0 ; i<len ; i++) {
      p[k] = 0 ;
      fstr[k++] = '#';
      p[k] = 0 ;
      fstr[k++] = str[i] ;
   }
   fstr[k++] = '#' ;
   fstr[k++] = '$' ;
   len = len*2 + 2 ;
}
int manchers() {
   int r = 0, c = 0, miror ;
   int mx = 0 ;
   for(int i=1 ; i<len ; i++) {
      miror = 2*c - i ;
      p[i] = r > i ? min(r-i,p[miror]) : 0 ;
      while(fstr[i+1+p[i]] == fstr[i-1-p[i]])
         p[i]++;
      if(i+p[i] > r) {
         c = i ;
         r = i+p[i] ;
      }
      mx = max(mx,p[i]);
   }
   return mx;
}
```

72. Extended Palindromic Tree

```
struct node
{
   int len,suff,in[29],st,en,diff,smart;
};
#define mx 400000
node tree[mx];
int ptr=2,curnode=1;
char str[mx];
int rev[mx],dp[mx][3],dplink[mx][3];
```

```
void makepal(int indx)
{
   int temp=curnode;
   while(1)
   {
      int len=tree[temp].len;
      if(indx-len>=1&&str[indx]==str[indx-1-len])
         break;
      temp=tree[temp].suff;
   }
   if(tree[temp].in[str[indx]-'a'])
   {
      curnode=tree[temp].in[str[indx]-'a'];
      rev[indx]=curnode;
      return ;
   }
   tree[temp].in[str[indx]-'a']=++ptr;
   dplink[ptr][0]=dplink[ptr][1]=mx;
   tree[ptr].len=tree[temp].len+2;
   tree[ptr].st=indx-tree[ptr].len+1;
   tree[ptr].en=indx;
   curnode=ptr;
   rev[indx]=curnode;
   temp=tree[temp].suff;
   if(tree[ptr].len==1)
   {
      tree[ptr].suff=2;
      tree[ptr].diff=1;
      tree[ptr].smart=0;
      return ;
   }
   while(1)
   {
      int len=tree[temp].len;
      if(indx-len>=1&&str[indx]==str[indx-1-len])
         break;
      temp=tree[temp].suff;
   }
   int s=tree[curnode].suff=tree[temp].in[str[indx]-'a'];
   tree[curnode].diff=tree[curnode].len-tree[s].len;
   if(tree[curnode].diff!=tree[s].diff)
      tree[curnode].smart=s;
   else
      tree[curnode].smart=tree[s].smart;
}
int sz(int v)
{
   return tree[v].len;
}
int main()
{
   tree[1].diff=0;
   tree[1].len=-1;
   tree[1].suff=1;
   tree[2].diff=0;
   tree[2].suff=1;
   tree[2].len=0;
   scanf("%s",str+1);
   int i;
   for(int i=1; str[i]; i++)
   {
      makepal(i);
```

```
    dp[i][0]=mx;
    dp[i][1]=mx;
    dplink[i][0]=mx;
    dplink[i][1]=mx;
  }
  dp[0][0]=0;
  dp[0][1]=mx;
  for(i=1; str[i]; i++)
  {
    for(int last=rev[i]; tree[last].len; last=tree[last].smart)
    {
      int s=tree[last].smart;
      int f=tree[last].suff;
      dplink[last][0]=mx;
      dplink[last][1]=mx;
      dplink[last][0]=min(dplink[last][0],dp[i-(sz(s)+tree[last].diff)][1]);
      dplink[last][1]=min(dplink[last][1],dp[i-(sz(s)+tree[last].diff)][0]);
      //  if(i==4)cout<<dplink[last][1]<<" "<<" "<<last<<" "<<dp[(i-
(sz(s)+tree[last].diff))][1]<<endl;
      if(tree[last].diff==tree[tree[last].suff].diff)
      {
        dplink[last][0]=min(dplink[last][0],dplink[f][0]);
        dplink[last][1]=min(dplink[last][1],dplink[f][1]);
      }
      //   if(i==4)cout<<dplink[last][1]<<" "<<" "<<f<<" "<<dp[(i-
(sz(s)+tree[last].diff))][1]<<endl;
      dp[i][0]=min(dp[i][0],dplink[last][0]+1);
      dp[i][1]=min(dp[i][1],dplink[last][1]+1);
    }
  }
  for(int i=1; str[i]; i++)
  {
    if(dp[i][1]>=mx)
      dp[i][1]=-1;
    if(dp[i][0]>=mx)
      dp[i][0]=-2;
    printf("%d %d\n",dp[i][1],dp[i][0]);
  }
}
```

73. Z-Algo

```
const int NX = 1e5 + 10  ; // string size
char text[NX] ;
int Z[NX];

void Z_Algorithm()
{
   int position , starting_point , ending_point ;
   int sz = strlen( text );
   Z[0] = sz ; // always ;
   for ( position = 1 , starting_point = 0 , ending_point = 0 ; position < sz ;
position++ )
   {
      if( position <= ending_point ) Z[position] = min( ending_point -
position + 1 , Z[position-starting_point] );
      while( position + Z[position] < sz && text[Z[position]] == text[
position + Z[position] ] ) ++Z[position] ;
      if ( position + Z[position] - 1  > ending_point ) // need to update
      starting_point = position , ending_point = position + Z[position] - 1 ;
   }
}
```

74. Palindromic Tree

```
struct node
{
   int len,suff,in[29],st,en;
};
node tree[200000];
int ptr,curnode;
char str[200000];
void makepal(int indx)
{
   int temp=curnode;
   while(1)
   {
      int len=tree[temp].len;
      if(indx-len>=1&&str[indx]==str[indx-1-len])
         break;
      temp=tree[temp].suff;
   }
   if(tree[temp].in[str[indx]-'a'])
   {
      curnode=tree[temp].in[str[indx]-'a'];
      return ;
   }
   tree[temp].in[str[indx]-'a']=++ptr;
   tree[ptr].len=tree[temp].len+2;
   tree[ptr].st=indx-tree[ptr].len+1;
   tree[ptr].en=indx;
   curnode=ptr;
   temp=tree[temp].suff;
   if(tree[ptr].len==1)
   {
      tree[ptr].suff=2;
      return ;
   }
   while(1)
   {
      int len=tree[temp].len;
      if(indx-len>=1&&str[indx]==str[indx-1-len])
         break;
      temp=tree[temp].suff;
   }
   tree[curnode].suff=tree[temp].in[str[indx]-'a'];
}
int main()
{
   node root1,root2;
   for(int i=1; i<=200000; i++)
   {
      for(int j=0; j<26; j++)
      {
         tree[i].in[j]=0;
         root1.in[j]=0;
         root2.in[j]=0;
      }
   }
   root1.len=-1;
   root1.suff=1;
   root2.len=0;
   root2.suff=1;
   tree[1]=root1;
   tree[2]=root2;
```

```
curnode=1;
ptr=2;
scanf("%s",str);
for(int i=0; str[i]; i++)
{
        makepal(i);
        printf("%d ",ptr-2);
    }
}
```

Dynamic Programming Optimization

75. Notes

Convex Hull Optimization1	$dp[i] = min_{j<ij}\{ dp[k] + b[k] * a[j]\}$	$b[k] \geq b[k+1]$ ~~optionally~~ $a[j] \leq a[j+1]$	$O(n^2)$	$O(n)$
Convex Hull Optimization2	$dp[i][j] = min_{k<j}\{ dp[i-1][k] + b[k] * a[j]\}$		$O(kn^2)$	$O(kn)$
Divide and Conquer Optimization	$dp[i][j] = min_{k<j}\{dp[i-1][k] + C[k][j]\}$	$A[i][j] \leq A[i][j+1]$	$O(kn^2)$	$O(knlogn)$
Knuth Optimization	$dp[i][j] = min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$	$A[i,j-1] \leq A[i,j] \leq A[i+1,j]$	$O(n^3)$	$O(n^2)$

Notes:

- $A[i][j]$ — the smallest k that gives optimal answer, for example in $dp[i][j] = dp[i-1][k] + C[k][j]$
- $C[i][j]$ — some given cost function
- We can generalize a bit in the following way: $dp[i] = min_{j<i}\{F[j] + b[j] * a[i]\}$, where $F[j]$ is computed from $dp[j]$ in constant time.
- It looks like **Convex Hull Optimization2** is a special case of **Divide and Conquer Optimization**.
- It is claimed (in the references) that **Knuth Optimization** is applicable if $C[i][j]$ satisfies the following 2 conditions:
  quadrangle inequality: $C[a][c] + C[b][d] \leq C[a][d] + C[b][c],\ a \leq b \leq c \leq d$
  monotonicity: $C[b][c] \leq C[a][d],\ a \leq b \leq c \leq d$

76. CovexHull Trick 1D

```
struct cline {
    ll M, C;
    cline() {}
    cline(ll m, ll c): M(m), C(c) {}
```

```
};
//pointer=0,last=0 should be made initially
cline line[MX]; //y=mx+c we need only m(slope) and c(constant)
//Returns true if either line l1 or line l3 is always better than line l2

bool bad(const cline & l1,const cline & l2,const cline & l3) {
   /*intersection(l1,l2) has x-coordinate (c1-c2)/(m2-m1)
   intersection(l1,l3) has x-coordinate (c1-c3)/(m3-m1)
   set the former greater than the latter, and cross-multiply to
   eliminate division*/
   //if the query x values is non-decreasing (reverse(> sign) for vice verse)
   return (l3.C-l1.C)*(l1.M-l2.M)<=(l2.C-l1.C)*(l1.M-l3.M);}
//Adding should be done serially
//If we want minimum y coordinate(value) then maximum valued m
should be inserted first
//If we want maximum y coordinate(value) then minimum valued m
should be inserted first
void add(cline l,int &last) {
   //First, let's add it to the end
   line[last++]=l;
   //If the penultimate is now made irrelevant between the
antepenultimate
   //and the ultimate, remove it. Repeat as many times as necessary
   //in short convex hull main convex hull tecnique is applied here
   while(last>=3&&bad(line[last-3],line[last-2],line[last-1])) {
      line[last-2]=line[last-1];
      last--;
   }
}
//Returns the minimum y-coordinate of any intersection between a given
vertical
//line(x) and the lower/upper envelope(pointer)
//This can only be applied if the query of vertical line(x) is already sorted
//works better if number of query is huge
long long query(long long x,int &pointer,int last) {
   //If we removed what was the best line for the previous query, then
the
   //newly inserted line is now the best for that query
   if (pointer>=last)
      pointer=last-1;
   //Any better line must be to the right, since query values are
   //non-decreasing
   // Min Value wanted... (reverse(> sign) for max value)
   while (pointer<last-1 &&line[pointer+1].M*x+line[pointer+1].C
<=line[pointer].M*x+line[pointer].C)
      pointer++;
   return line[pointer].M*x+line[pointer].C;
}
//for any kind of query(sorted or not) it can be used
//it works because of the hill property
//works better if number of query is few
long long bs(int st,int end,long long x,int last) {
   int mid=(st+end)/2;
// Min Value wanted... (reverse(> sign) for max value if(mid+1<last &&
line[mid+1].M*x+line[mid+1].C <line[mid].M*x+line[mid].C)
         return bs(mid+1,end,x,last);
   // Min Value wanted... (reverse(> sign) for max value)
   if(mid-1>=0 && line[mid-1].M*x+line[mid-
1].C<line[mid].M*x+line[mid].C)
return bs(st,mid-1,x,last);
   return line[mid].M*x+line[mid].C;
}
```

```
int main() {
   int last = 0, pointer = 0;
   return 0;
}
```

77. Covexhull Trick 2D

```
ll dp[MX][2];
ll func(int n, int p) {
   ll CostOfPartition;
   for(int i = 1; i <= n; i++)
      dp[i][1] = CostOfPartition;
   for(int k = 2; k <= p; k++) {
      int last = 0, pointer = 0;
      int cur = k&1;       int prv = (k-1)&1;
      for(int i = k; i <= n; i++) {
         ll M, C, CC, X;
           // M = slope of line    C = line constant factor
         add(cline(M,C),last);
           // X = value of query   CC = Extra cost for this partition
         dp[i][cur] = query(X,pointer,last)+CC;
      }
   }
   return dp[n][p&1];
}
```

78. Divide and Conquer (Jubair)

```
int
sum[4200][4200],dp[4200][4200],ara[4200][4200],ddsum[4200][4200];
void dp_func(int row,int st,int en,int l,int r)
{
   if(st>en)
      return ;
   int mid=(st+en)/2;
   int best=l;
   dp[row][mid]=2147483647;
   for(int i=l; i<=min(mid-1,r); i++)
   {
      int k=i+1;
      int d=ddsum[k][mid]+dp[row-1][i];
      if(d<dp[row][mid])
      {
         dp[row][mid]=d;
         best=i;
      }
   }
   if(st==en)
      return ;
   dp_func(row,st,mid-1,l,best);
   dp_func(row,mid+1,en,best,r);
}
int main()
{
   int i,j,k,l,m,n;
   char str[400000];
   scanf("%d%d",&n,&m);
```

```
    getchar();
    char ch,ch2;
    for(i=1; i<=n; i++)
    {
        gets(str);
        j=1;
        for(k=0; str[k]; k+=2)
        {
            ara[i][j]=str[k]-48;
            sum[i][j]=sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1]+ara[i][j];
            j++;
        }
    }
    for(i=1; i<=n; i++)
    {
        for(j=i; j<=n; j++)
        {
            ddsum[i][j]=sum[j][j]-sum[i-1][j]-sum[j][i-1]+sum[i-1][i-1];
        }
    }
    for(i=1; i<=n; i++)
    {
        dp[1][i]=ddsum[1][i];
    }
    for(i=2; i<=m; i++)
    {
        dp_func(i,1,n,0,n-1);
    }
    printf("%d\n",dp[m][n]/2);
}
```

79. Divide and Conquer (TeamX)

```
int
sum[4200][4200],dp[4200][4200],ara[4200][4200],ddsum[4200][4200];
void dp_func(int row,int st,int en,int l,int r)
{
    if(st>en)
        return ;
    int mid=(st+en)/2;
    int best=l;
    dp[row][mid]=2147483647;
    for(int i=l; i<=min(mid-1,r); i++)
    {
        int k=i+1;
        int d=ddsum[k][mid]+dp[row-1][i];
        if(d<dp[row][mid])
        {
            dp[row][mid]=d;
            best=i;
        }
    }
    if(st==en)
        return ;
    dp_func(row,st,mid-1,l,best);
    dp_func(row,mid+1,en,best,r);
}
int main()
{
    int i,j,k,l,m,n;
    char str[400000];
    scanf("%d%d",&n,&m);
```

```
    getchar();
    char ch,ch2;
    for(i=1; i<=n; i++)
    {
        gets(str);
        j=1;
        for(k=0; str[k]; k+=2)
        {
            ara[i][j]=str[k]-48;
            sum[i][j]=sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1]+ara[i][j];
            j++;
        }
    }
    for(i=1; i<=n; i++)
    {
        for(j=i; j<=n; j++)
        {
            ddsum[i][j]=sum[j][j]-sum[i-1][j]-sum[j][i-1]+sum[i-1][i-1];
        }
    }
    for(i=1; i<=n; i++)
    {
        dp[1][i]=ddsum[1][i];
    }
    for(i=2; i<=m; i++)
    {
        dp_func(i,1,n,0,n-1);
    }
    printf("%d\n",dp[m][n]/2);
}
```

80. Knuth Optimization 1

```
int sum[MAX][MAX];
int dp[MAX][MAX];
int opt[MAX][MAX];
inline int cost(int u, int v) {
    return sum[v][v] - sum[v][u] - sum[u][v] + sum[u][u];
}
int main() {
    for (int i = 1; i <= N; ++i)
        dp[1][i] = cost(0, i), opt[1][i] = 1;
    for (int i = 2; i <= K; ++i)
        for (int j = N; j; --j) {
            dp[i][j] = inf;           opt[i][N + 1] = N;
            for (int k = opt[i - 1][j]; k <= opt[i][j + 1]; ++k)
                if (dp[i][j] > dp[i - 1][k] + cost(k, j)) {
                    dp[i][j] = dp[i - 1][k] + cost(k, j);
                    opt[i][j] = k;
                }
        }
    return 0;
}
```

81. Knuth Optimization 2

Knuths optimization works for optimization over substrings for which optimal middle point depends monotonously on the end points. Let mid[l,r] be the first middle point for (l,r) substring which gives optimal result. It can be proven that mid[l,r-1] <= mid[l,r] <= mid[l+1,r] - this means monotonicity of mid by l and r.

Applying this optimization reduces time complexity from O(k^3) to O(k^2) because with fixed s (substring length) we have m_right(l) = mid[l+1][r] = m_left(l+1). That's why nested l and m loops require not more than 2k iterations overall.

```
for (int s = 0; s<=k; s++)                  //s - length(size) of substring
   for (int l = 0; l+s<=k; l++) {            //l - left point
     int r = l + s;                        //r - right point
     if (s < 2) {
       res[l][r] = 0;                      //DP base - nothing to break
       mid[l][r] = l;                      //mid is equal to left border
       continue;
     }
     int mleft = mid[l][r-1]; //Knuth's trick: getting bounds on m
     int mright = mid[l+1][r];
     res[l][r] = 1000000000000000000LL;
     for (int m = mleft; m<=mright; m++) {     //iterating for m in the
bounds only
       int64 tres = res[l][m] + res[m][r] + (x[r]-x[l]);
       if (res[l][r] > tres) {                //relax current solution
         res[l][r] = tres;         mid[l][r] = m;
       }
     }
   }
int64 answer = res[0][k];
```

82. Knuth Optimization 3

```
int dp[1003][1003],best[1003][1003],ara[1003];
int dp_func(int st,int en)
{
   if(en-st==1)
   {
      best[st][en]=st;
      return dp[st][en]=0;
   }
   if(dp[st][en]!=-1)
      return dp[st][en];
   dp_func(st,en-1);
   dp_func(st+1,en);
   int f=best[st][en-1];
   int e=best[st+1][en];
   //  if(st==0&&en==4)cout<<f<<" "<<e<<endl;
   int d=ara[en]-ara[st];
   int ret=2147483647,bst=st;
   for(int i=max(st+1,f); i<=min(en-1,e); i++)
   {
      int g=dp_func(st,i)+dp_func(i,en)+d;
      if(g<ret)
      {
         ret=g;
         bst=i;
      }
   }
   best[st][en]=bst;
   return dp[st][en]=ret;
}
int main()
{
   int n,m;
   while(scanf("%d%d",&n,&m)==2)
   {
```

```
    memset(dp,-1,sizeof dp);
    for(int i=1; i<=m; i++)
      scanf("%d",&ara[i]);
    ara[m+1]=n;
    int ans=dp_func(0,m+1);
    printf("%d\n",ans);
  }
}
```

83. SOS DP

```
const int LN = 20 ;
int dp[(1<<LN)+7] ;

void rec() {
  /** we must initialize dp array with value based on problem.
    Only one inner loop will be activated base on problem  **/
  /** actual dp state we write optimize version .
    if(mask & (1<<i))
            dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
      else
            dp[mask][i] = dp[mask][i-1];
  **/

  for(int i=0 ; i<LN ; i++) {
    for(int j=0 ; j<(1<<LN) ; j++) {
      /**
        this loop is used if we want j&i == i
        specifically mask&x == x ; here mask is j
      **/
      if(j&(1<<i))
        dp[j] += dp[j^(1<<i)] ;
    }

    for(int j=(1<<LN)-1 ; j>=0 ; j--) {
      /**
        this loop is used if we want mask&x == mask
      **/

      if((j&(1<<i)) == 0)
        dp[j] += dp[j^(1<<i)] ;
    }
  }
}
```

84. Dynamic Convex Hull

```
typedef long long ll;
typedef long double float128;

const ll is_query = -(1LL<<62), inf = 1e18;

struct Line
{
  ll m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const
  {
    if (rhs.b != is_query)
      return m < rhs.m;
    const Line* s = succ();
    if (!s)
      return 0;
    ll x = rhs.m;
```

```
        return b - s->b < (s->m - m) * x;
    }
};

struct HullDynamic : public multiset<Line>//will maintain lower hull for
maximum
{
    bool bad(iterator y)
    {
        auto z = next(y);
        if (y == begin())
        {
            if (z == end())
                return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end())
            return y->m == x->m && y->b <= x->b;
        return (float128)(x->b - y->b)*(z->m - y->m) >= (float128)(y->b - z-
>b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b)
    {
        auto y = insert({ m, b });//for maxi
        //   auto y = insert({ -m, -b });// for here for minimum
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y))
        {
            erase(y);
            return;
        }
        while (next(y) != end() && bad(next(y)))
            erase(next(y));
        while (y != begin() && bad(prev(y)))
            erase(prev(y));
    }
    //for query, Line can't be empty
    ll eval(ll x)
    {
        auto l = *lower_bound((Line)
        {
            x, is_query
        });
        return (l.m * x + l.b);//here for maxi
        //   return -(l.m * x + l.b);////  here for minimum
    }
};
```

85. CHT more Dynamic

```
const int mod=1e9+7;
const int N=5e4+9;
const ld eps=1e-9;
const ld PI=acos(-1.0);
//ll gcd(ll a,ll b){while(b){ll x=a%b;a=b;b=x;}return a;}
//ll lcm(ll a,ll b){return a/gcd(a,b)*b;}
//ll qpow(ll n,ll k) {ll ans=1;assert(k>=0);n%=mod;while(k>0){if(k&1)
ans=(ans*n)%mod;n=(n*n)%mod;k>>=1;}return ans%mod;}

const ll nsz=5e4+9;//maximum number of lines
ll msz;//make it 0 for restarting the CHT
ll outside = nsz-1;
```

```
ll M[nsz], B[nsz]; // y = M*X + B formatted lines, must be sorted in
advanced by M //clear M, B for test cases, make qptr = 0
bool bad(int l1, int l2, int l3, bool lowerPart = 1) // returns true if l1-l3 line
is better than l2
{
   /*
   intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
   intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
   */
 //   cout << (B[l3]-B[l1])*(M[l1]-M[l2]) << " " << (B[l2]-B[l1])*(M[l1]-M[l3])
<< endl;
   if (lowerPart == 1)
      return 1.00*(B[l3]-B[l1])*(M[l1]-M[l2]) <= 1.00*(B[l2]-B[l1])*(M[l1]-
M[l3]);
   else return 1.00*(B[l3]-B[l1])*(M[l1]-M[l2]) >= 1.00*(B[l2]-B[l1])*(M[l1]-
M[l3]);
}
struct data //information to undo change in CHT
{
   ll m, b, pos;
   data(ll _m = 0, ll _b = 0, ll _pos = 0)
   {
      m = _m, b = _b, pos = _pos;
   }
};
data add(ll m, ll b, bool lowerPart = 1)
{
// lowerPart is called upper hull. For m decreasing, this creates lower
part, but if m increasing, it does reverse
// lower part is needed for finding minimum, upper part for maximum
   M[outside] = m, B[outside] = b;
   while (msz >= 2 && bad(msz-2, msz-1, outside, lowerPart))
   {
      msz--;
   }
   data temp(M[msz], B[msz], msz);
   M[msz] = m;
   B[msz] = b;
   msz++;
   return temp;
}
ll query(ll x, bool findMin = 1) //online query
{
   int lo = 0, hi = msz - 1;
   ll ans = LLONG_MAX;
   if (findMin)
      ans = -LLONG_MAX;
   while(lo <= hi)
   {
      int diff = (hi-lo)/3;
      int mid1 = lo + diff;
      int mid2 = hi - diff;
      ll y1 = M[mid1]*x + B[mid1], y2 = M[mid2]*x + B[mid2];
      if(y1 <= y2)
      {
         ans = y1;
         if (findMin)
            hi = mid2 - 1;
         else lo = mid1 + 1;
      }
      else
      {
```

```
            ans = y2;
            if (findMin)
               lo = mid1 + 1;
            else hi = mid2 - 1;
         }
      }
      return ans;
}
ll sum[N],val[N],a,b,ans;
vll g[N];
void dfs(ll u,ll pre=0)
{
   sum[u]=val[u];
   for(auto v:g[u]){
      if(v==pre) continue;
      dfs(v,u);
      sum[u]+=sum[v];
   }
}
void yo(ll u,ll pre=0)
{
   bool leaf=1;
   for(auto v:g[u]){
      if(v==pre) continue;
      leaf=0;
      ll res=query(sum[v])+a*sum[v]*sum[v]+b;
      ans=min(ans,res+a*sum[v]*sum[v]+b);
      ll prvsz=msz;
      data undo=add(-2*a*sum[v],a*sum[v]*sum[v]+res,0);
      yo(v,u);
      msz=prvsz;
      M[undo.pos]=undo.m;
      B[undo.pos]=undo.b;
   }
   if(leaf) ans=min(ans,query(0)+b);
}
int main()
{
   fast;
   ll i,j,k,n,m,t,u,v;
   cin>>t;
   while(t--){
      cin>>n>>a>>b;
      for(i=1;i<=n;i++) cin>>val[i];
      for(i=1;i<n;i++){
         cin>>u>>v;
         g[u].eb(v);
         g[v].eb(u);
      }
      dfs(1);
      ans=a*sum[1]*sum[1]+b;
      msz=0;
      add(-2*a*sum[1],a*sum[1]*sum[1]);
      yo(1);
      cout<<ans<<nl;
      mem(sum,0);
      for(i=1;i<=n;i++) g[i].clear();
   }
   return 0;
}
```

86. Connected Component DP

```
#include <bits/stdc++.h>
using namespace std;
template <class T> int size(const T &x) { return x.size(); }
#define rep(i,a,b) for (__typeof(a) i=(a); i<(b); ++i)
#define iter(it,c) for (__typeof((c).begin()) it = (c).begin(); it != (c).end();
++it)
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
typedef long long ll;
const int INF = 2147483647;

int arr[1010];
int n, l;

int mem[110][1010][2][110][2];
int mod = 1000000007;

ll dp(int at, int curl, int kl, int k, int kr) {

   // kl = 1 if there is a segment connected to the left border, 0 otherwise
   // kr = 1 if there is a segment connected to the right border, 0
otherwise
   // k is the number of segments in the middle

   int nxtl = curl;
   if (at > 0) {
      // add the penalty from the last element:
      nxtl += (kl+kr+2*k)*abs(arr[at]-arr[at-1]);
   }

   if (nxtl > l) return 0;

   if (k < 0) return 0;

   if (at == n-1) {
      return k == 0 ? 1 : 0;
   }
   if (mem[at][curl][kl][k][kr] != -1)
      return mem[at][curl][kl][k][kr];

   ll res = 0;

   res += dp(at+1, nxtl, 1, k, kr); // connect to left segment
   res += dp(at+1, nxtl, 1, k-1, kr)*k; // connect to left segment, and join to
some middle segment

   res += dp(at+1, nxtl, kl, k, 1); // connect to right segment
   res += dp(at+1, nxtl, kl, k-1, 1)*k; // connect to right segment, and join
to some middle segment

   res += dp(at+1, nxtl, kl, k+1, kr); // new segment
   res += dp(at+1, nxtl, kl, k, kr)*k*2; // connect to some middle segment
   res += dp(at+1, nxtl, kl, k-1, kr)*k*(k-1); // join two middle segments

   return mem[at][curl][kl][k][kr] = res % mod;
}
int main()
{
   memset(mem,-1,sizeof(mem));
   cin >> n >> l;
   rep(i,0,n) cin >> arr[i];
   sort(arr,arr+n);
   cout << dp(0, 0, 0, 0, 0) << endl;
   return 0;
}
```

Matrix Related Algorithm

87. Guass Elimination

```
//a is the total matrix, last column is the constant matrix and other
columns are coefficient matrix
//final ans is stored is ans matrix

int gauss ( vector < vector < double > > a, vector < double > & ans ) {
   int n = ( int ) a. size ( ) ;
   int m = ( int ) a [ 0 ] . size ( ) - 1 ;

   vector < int > where ( m, - 1 ) ;
   for ( int col = 0, row = 0 ; col < m && row < n ; ++ col ) {
      int sel = row ;
      for ( int i = row ; i < n ; ++ i )
         if ( abs ( a [ i ] [ col ] ) > abs ( a [ sel ] [ col ] ) ) //maxvalued row for
this column
            sel = i ;
      if ( abs ( a [ sel ] [ col ] ) < ERR )
         continue ;
      for ( int i = col ; i <= m ; ++ i )
         swap ( a [ sel ] [ i ], a [ row ] [ i ] ) ;  //swap the rows
      where [ col ] = row ;

      for ( int i = 0 ; i < n ; ++ i )
         if ( i != row ) {
            double c = a [ i ] [ col ] / a [ row ] [ col ] ;
            for ( int j = col ; j <= m ; ++ j )
               a [ i ] [ j ] -= a [ row ] [ j ] * c ;
         }
      ++ row ;
   }
   debug("::::::::::::::::::::::::::::");
   for(int i=0; i<n; i++) {
      for(int j=0; j<=m; j++) {
         printf("%5.2f ",a[i][j]);
      }
      printf("\n");
   }
   debug("::::::::::::::::::::::::::::");

   ans. assign ( m, 0 ) ;
   for ( int i = 0 ; i < m ; ++ i )
      if ( where [ i ] != - 1 )
         ans [ i ] = a [ where [ i ] ] [ m ] / a [ where [ i ] ] [ i ] ;
   //checking right
   for ( int i = 0 ; i < n ; ++ i ) {
      debug("***",where[i]);
      double sum = 0 ;
      for ( int j = 0 ; j < m ; ++ j )
         sum += ans [ j ] * a [ i ] [ j ] ;
      if ( abs ( sum - a [ i ] [ m ] ) > ERR ) //no solution
         return 0 ;
   }

   for ( int i = 0 ; i < m ; ++ i )
      if ( where [ i ]== - 1 ) //infinite solution
         return INF;
   return 1 ; //unique solution
}
```

```
int main() {
   int n,m;
   while(scanf("%d",&n)==1) {
      vector<vector<double> > mat(n);
      vector<double> ans;
      double v;
      for(int i=0; i<n; i++) {
         for(int j=0; j<n+1; j++) {
            scanf("%lf",&v);
            mat[i].push_back(v);
         }
      }
      debug(gauss(mat,ans));
      for(int i=0; i<n; i++) debug(ans[i]);
   }
   return 0;
}

/**
3
1 2 1 3
2 0 1 1
0 1 1 2
*/
```

88. Guass Elimination(row order)

//a is the total matrix, last column is the constant matrix and other columns are coefficient matrix
//final ans is stored is ans matrix
//row order is kept and assigned the given (intended) value to first row then second row ans so on

```
long long gauss ( vector < vector < long long > > a, vector < long long > &
ans, long long mod) {
   int n = ( int ) a. size ( ) ;
   int m = ( int ) a [ 0 ] . size ( ) - 1 ;

   vector < int > where ( n, - 1 ) ;
   for ( int col = 0, row = 0 ; col < m && row < n ; ++ row ) {
      int sel = col ;
      for ( int i = col ; i < m ; ++ i )
         if ( abs ( a [ row ] [ i ] ) > abs ( a [ row ] [ sel ] ) )
            sel = i ;
      if ( abs ( a [ row ] [ sel ] ) == 0 )
         continue ;
      for ( int i = 0 ; i < n ; ++ i )
         swap ( a [ i ] [ col ], a [ i ] [ sel ] ) ;
      where [ row ] = col ;
      //print3(row,col,a[row][col]);

      for ( int i = 0 ; i < n ; ++ i )
         if ( i != row ) {
            long long c = a [ row ] [ col ];
            long long d = a [ i ][ col ];
            for ( int j = col ; j <= m ; ++ j ) {
               a [ i ] [ j ] = (c*a[i][j]-d*a[row][j])%mod ;
               a [i][j]=(a[i][j]+mod)%mod;
            }
         }
      ++ col ;
```

```
    }
    ans. assign ( m, 0 ) ;
    for ( int i = 0 ; i < n ; ++ i )
      if ( where [ i ] != - 1 )
        ans [ where[i] ] = (a [  i  ] [ m ]*
                    bigmod( a [ i ] [ where[i] ],mod-2,mod))%mod ;
    for ( int i = 0 ; i < n ; ++ i ) {
      long long sum = 0 ;
      for ( int j = 0 ; j < m ; ++ j ) {
        sum += (ans [ j ] * a [ i ] [ j ])%mod ;
        sum %= mod;
      }
      if ( abs ( sum - a [ i ] [ m ] ) != 0 ) //no solution
        return 0 ;
    }

    long long totalans=1;
    for ( int i = 0 ; i < m ; ++ i )
      if ( where [ i ]== - 1 ) //use mod if necessary
        totalans=(totalans* mod)%1000000007;

    return totalans ;
}
```

89. Guass Elimination(Modular)

```
//a is the total matrix, last column is the constant matrix and other
columns are coefficient matrix
//final ans is stored in ans matrix
long long gauss ( vector < vector < long long > > a, vector < long long > &
ans, long long mod) {
    int n = ( int ) a. size ( ) ;
    int m = ( int ) a [ 0 ] . size ( ) - 1 ;

    vector < int > where ( m, - 1 ) ;
    for ( int col = 0, row = 0 ; col < m && row < n ; ++ col ) {
      int sel = row ;
      for ( int i = row ; i < n ; ++ i )
        if ( abs ( a [ i ] [ col ] ) > abs ( a [ sel ] [ col ] ) )
          sel = i ;
      if ( abs ( a [ sel ] [ col ] ) == 0 )
        continue ;
      for ( int i = col ; i <= m ; ++ i )
        swap ( a [ sel ] [ i ], a [ row ] [ i ] ) ;
      where [ col ] = row ;
      //print3(row,col,a[row][col]);

      for ( int i = 0 ; i < n ; ++ i )
        if ( i != row ) {
          long long c = a [ row ] [ col ];
          long long d = a [ i ][ col ];
          for ( int j = col ; j <= m ; ++ j ) {
            a [ i ] [ j ] = (c*a[i][j]-d*a[row][j])%mod ;
            a [i][j]=(a[i][j]+mod)%mod;
            //print3(i,j,a[i][j]);
          }
        }
      //cout<<endl;
      ++ row ;
    }
    ans. assign ( m, 0 ) ;
    for ( int i = 0 ; i < m ; ++ i )
      if ( where [ i ] != - 1 )
```

```
        ans [ i ] = (a [ where [ i ] ] [ m ] * bigmod( a [ where [ i ] ] [ i ],mod-
2,mod))%mod ;
   for ( int i = 0 ; i < n ; ++ i ) {
      long long sum = 0 ;
      for ( int j = 0 ; j < m ; ++ j ) {
         sum += (ans [ j ] * a [ i ] [ j ])%mod ;
         sum %= mod;
      }
      if ( abs ( sum - a [ i ] [ m ] ) != 0 ) //no solution
         return 0 ;
   }

   long long totalans=1;
   for ( int i = 0 ; i < m ; ++ i )
      if ( where [ i ]== - 1 ) //use mod if necessary
         totalans=(totalans* mod)%1000000007;

   return totalans ;
}
```

90. Guass Elimination(Mod 2)

```
//a is the total matrix, last column is the constant matrix and other
columns are coefficient matrix
//final ans is stored in ans matrix
//complexity (n^3)/64

long long gauss ( vector < vector < long long > > a, vector < long long > &
ans,int sz) { //sz=number of variables+1
   int n = ( int ) a. size ( ) ;
   int m = sz-1;
   //print2(n,m);
   vector < int > where ( m, - 1 ) ;
   for ( int col = 0, row = 0 ; col < m && row < n ; ++ col ) {
      int sel = row ;
      //print1(a[row][col]);
      for ( int i = row ; i < n ; ++ i )
         if (((a[i][col/64])&(1LL<<(col%64))) >
((a[sel][col/64])&(1LL<<(col%64))))
            sel = i ;
      if ( ((a[sel][col/64])&(1LL<<(col%64)))==0 )
         continue ;
      for ( int i = col/64 ; i <= m/64 ; ++ i )
         swap ( a [ sel ] [ i ], a [ row ] [ i ] ) ;
      where [ col ] = row ;
      //print3(row,col,a[row][col]);

      for ( int i = 0 ; i < n ; ++ i )
         if ( i != row ) {
            if((a[i][col/64])&(1LL<<(col%64))) //if set
               for ( int j = col/64 ; j <= m/64 ; ++ j ) {
                  a [ i ][ j ] ^= a[row][j];

               }
         }
      ++ row ;
   }
   ans. assign ( m, 0 ) ;
   for ( int i = 0 ; i < m ; ++ i )
      if ( where [ i ] != - 1 ) {
         ans [ i ] =  (a [ where [ i ] ] [ m/64 ]& (1LL<<(m%64)));
         if(ans[i]) ans[i]=1;
      }
```

```
    for ( int i = 0 ; i < n ; ++ i ) {
        bool sum = 0 ;
        for ( int j = 0 ; j < m ; ++ j ) {
            int gun=(a [ i ] [ j/64 ]& (1LL<<(j%64)));
            if(gun) gun=1;
            sum += ans [ j ] *gun;
        }
        if( sum!= (bool)(a[i][m/64]&(1LL<<(m%64)) ))
            return 0;
    }

    long long totalans=1;
    for ( int i = 0 ; i <= m ; ++ i )
        if ( where [ i ]== - 1 ) //use mod if necessary
            totalans=(totalans* 2)%1000000007;

    return totalans;
}
int main() {
    int t,cas=0;
    cin>>t;
    while(t--) {
        int n,m;
        cin>>n>>m;
        mem(grid,0);
        int i,j;
        for(i=1; i<=m; i++) {
            int k;
            scanf("%d",&k);
            int light;
            while(k--) {
                scanf("%d",&light);
                grid[light][i]=1;
            }
        }
        int q;
        cin>>q;
        csprnt;
        while(q--) {
            vector<long long>ans;
            vector< vector<long long> > a;
            for(i=1; i<=n; i++) {
                int state;
                scanf("%d",&state);
                vector<long long>tem;
                long long temval=0;
                for(j=1; j<=m; j++)
                    temval+=((long long)grid[i][j]<<(j-1));
                temval+=(long long)state<<m;
                tem.pb(temval);
                a.pb(tem);
            }
            printf("%I64d\n",gauss(a,ans,m+1));
        }
    }
    return 0;
}
```

91. Determinant

//a is the total matrix, last column is the constant matrix and other columns are coefficient matrix
//final ans is stored is ans matrix

```
int det (vector < vector < double > > a) {  //determinant of a square matrix
   int n=( int ) a. size ();
   int i, j, k, flg = 1;
   double ans=1.0,x;
   for (i = 0; i < n; i++) {
      int sol=i;
      for (j = i+1; j < n; j++)
         if (abs(a[j][i])>abs(a[sol][i]))
            sol=i;
      if(abs(a[i][sol])<ERR) return -1; //according to problem
      flg = !flg;
      for (k = i; k < n; k++)
         swap (a[i][k], a[j][k]);

      ans = ans * a[i][i];
      x=1.0/a[i][i];
      for (k = i+1; k < n; k++)
         a[i][k] = a[i][k] * x;
      for (j = i+1; j < n; j++)
         if (abs(a[j][i])<ERR) for (k = i+1; k < n; k++)
               a[j][k] = a[j][k] - a[i][k]*a[j][i];
   }
   if (flg) return ans;
   return -ans;
}
```

92. Determinant (modular)

//a is the total matrix, last column is the constant matrix and other columns are coefficient matrix
//final ans is stored is ans matrix

```
void Egcd (int a, int b, int &x, int &y) { //extended gcd
   if (b == 0) {
      x = 1, y = 0;
      return ;
   }
   Egcd (b, a%b, x, y);
   int tp = x;
   x = y;
   y = tp - a/b*y;
}

int det (vector < vector < long long > > a,int mod) {
//determinant of a square matrix
   int n=( int ) a. size ();
   int i, j, k, ans = 1, x, y, flg = 1;
   for (i = 0; i < n; i++) {
      if (a[i][i] == 0) {
         for (j = i+1; j < n; j++)
            if (a[j][i])
               break;
         if (j == n) return -1;
         flg = !flg;
         for (k = i; k < n; k++)
            swap (a[i][k], a[j][k]);
      }
      ans = ans * a[i][i] % mod;
      Egcd (a[i][i], mod, x, y); //inverse modulo
      x = (x%mod + mod) % mod;
      for (k = i+1; k < n; k++)
         a[i][k] = a[i][k] * x % mod;
```

```
    for (j = i+1; j < n; j++)
      if (a[j][i] != 0) for (k = i+1; k < n; k++)
          a[j][k] = ((a[j][k] - a[i][k]*a[j][i])%mod + mod) % mod;
  }
  if (flg) return ans;
  return mod-ans;
}
```

93. Mat Expo

```
const int MX = 40;
const int mod = 1000000007;

struct matrix{
  int row, col;
  int mat[MX+7][MX+7];

  matrix(int r, int c){
    memset(mat, 0, sizeof(mat));
    row = r, col = c;
  }
  void I3(){ /// convert to identity matrix
    for(int i=0; i<row; i++)
      mat[i][i] = 1;
  }
  void set_value(vector<vector<int>> arr){
    int i, j;
    for( i = 0; i<row; i++ )
      for( j = 0; j<col; j++ )
        mat[i][j] = arr[i][j];
  }
};

matrix multiply(matrix a, matrix b){
  if( a.col != b.row ) /// Multiplication not possible
    return matrix(-1, -1);
  matrix ans = matrix(a.row, b.col);
  int i, j, k;

  for( i=0; i<a.row; i++){
    for( j=0; j<b.col; j++){
      ll sum = 0;
      for( k=0; k<b.row; k++ )
        sum = (sum + ((ll)a.mat[i][k]*(ll)b.mat[k][j])%mod)%mod ;
      ans.mat[i][j] = sum;
    }
  }

  return ans;
}

matrix mat_power(matrix base, int n){
  matrix ans = matrix(base.row, base.col);
  ans.I3();

  while( n>0 ){
    if(n&1)
      ans = multiply(ans, base);
    base = multiply(base, base);
    n /= 2;
  }
```

```
    return ans;
}

int main(){
//    freopen("E:\\00.txt", "r", stdin);
    int t, cas, n, i, j, k;

    matrix mat = matrix(3, 3);

    vector<vector<int>> arr;
    arr.pb({4, 5, 2});
    arr.pb({4, 1, 4});
    arr.pb({3, 7, 5});

    mat.set_value(arr);

    matrix ans = mat_power(mat, 3);

    for( i = 0; i<ans.row; i++, puts("") )
        for( j = 0; j<ans.col; j++ )
            cout << ans.mat[i][j] << " ";

    return 0;
}
```

94. FFT(without modulo)

```
#include <bits/stdc++.h>
using namespace std;

#define pi acos(-1)

// nlogn complexity
// memory complexity 12n
/* application
   1. multiplying two arrays.
   2. multiplying two long(string) numbers.
*/
// i-th index mean coefficient of i-th power
typedef complex<long double> base;

void fft(vector<base> &a,bool invert) //invert=true means inverse FFT
{
    int n=(int)a.size();

    for(int i=1,j=0;i<n;++i)
    {
        int bit=n>>1;
        for(; j>=bit; bit>>=1) j-=bit;
        j+=bit;
        if(i<j) swap(a[i],a[j]);
    }

    for(int len=2;len<=n;len<<=1)
    {
        long double ang=2*pi/len*(invert?-1:1);
        base wlen(cos(ang),sin(ang));
        for(int i=0; i<n; i+=len)
        {
            base w(1);
            for(int j=0;j<len/2;++j)
            {
                base u=a[i+j],v=a[i+j+len/2]*w;
                a[i+j]=u+v;
                a[i+j+len/2]=u-v;
```

```
            w*=wlen;
          }
        }
    }
    if (invert)
      for (int i=0;i<n;++i) a[i]/=n;
    return ;
}

void multiply (vector<int> &a, vector<int> &b, vector<int> &res)
{
    vector<base> fa(a.begin(), a.end()),fb(b.begin(), b.end());
    size_t n = 1;
    while (n<max(a.size(),b.size())) n<<=1; //making it a power of 2
    n <<= 1; //making double size(2*n)
    fa.resize(n),fb.resize(n);

    fft(fa,false),fft(fb,false);
    for (size_t i=0;i<n;++i)
      fa[i]*=fb[i];
    fft(fa,true); //inverse fft

    res.resize(n);
    for (size_t i=0;i<n;++i)
      res[i]=int(fa[i].real()+0.5);
    return ;
}

void multiplyLongNum(vector<int> &a,vector<int> &b, vector < int > &
res )  //multiplying two long(string) numbers.(normalizing)
{
    reverse(a.begin(), a.end());
    reverse(b.begin(), b.end());
    multiply(a,b,res);
    int n=res.size();
    int carry = 0 ;
        for ( size_t i = 0 ; i < n ; ++ i ) {
            res [ i ] += carry ;
            carry = res [ i ] / 10 ;
            res [ i ] %= 10 ;
        }
        for(int i=res.size()-1;i>0;i--)
    {
      if(res[i]==0) res.pop_back();
      else break;
    }
    reverse(res.begin(), res.end());
}

int main()
{
    #ifdef MAHDI
//    Read;
//    Write;
    #endif // MAHDI
    vector<int> a{1,2,9};
    vector<int> b{7,0,3,8};
    vector<int> r;
    multiplyLongNum(a,b,r);
    for(int i=0;i<r.size();i++)
    {
      printf("%d",r[i]);
    }
    printf("\n");
    return 0;
}
```

95. FFT(without modulo+complexStructure)

```
#include <bits/stdc++.h>
using namespace std;

#define pi acos(-1.0)
// nlogn complexity
// memory complexity 12n
/* application
  1. multiplying two arrays.
  2. multiplying two long(string) numbers.
*/
// i-th index mean coefficient of i-th power


struct cmplx{
   long double r,i;
   inline cmplx(){r=i=0.0;}
   inline cmplx(long double x){r=x,i=0.0;}
   inline cmplx(long double x,long double y){r=x,i=y;}
   inline void operator+= (const cmplx &q){r+=q.r,i+=q.i;}
   inline void operator-= (const cmplx &q){r-=q.r,i-=q.i;}
   inline cmplx operator+ (const cmplx &q){
      return cmplx(r+q.r,i+q.i);
   }
   inline cmplx operator- (const cmplx &q){
      return cmplx(r-q.r,i-q.i);
   }
   inline cmplx operator* (const cmplx &q){
      return cmplx(r*q.r-i*q.i,r*q.i+i*q.r);
   }
};
typedef cmplx base;
void fft(vector<base> &a,bool invert) { //invert=true means inverse FFT
   int n=(int)a.size();

   for(int i=1,j=0; i<n; ++i) {
      int bit=n>>1;
      for(; j>=bit; bit>>=1) j-=bit;
      j+=bit;
      if(i<j) swap(a[i],a[j]);
   }

   for(int len=2; len<=n; len<<=1) {
      long double ang=2*pi/len*(invert?-1:1);
      base wlen(cos(ang),sin(ang));
      for(int i=0; i<n; i+=len) {
         base w(1);
         for(int j=0; j<len/2; ++j) {
            base u=a[i+j],v=a[i+j+len/2]*w;
            a[i+j]=u+v;
            a[i+j+len/2]=u-v;
            w=w*wlen;
         }
      }
   }
   if (invert)
      for (int i=0; i<n; ++i) a[i].r/=n;
   return ;
}
```

```
void multiply (vector<int> &a, vector<int> &b, vector<int> &r) {
   vector<base> fa(a.begin(), a.end()),fb(b.begin(), b.end());
   size_t n = 1;
   while (n<max(a.size(),b.size())) n<<=1; //making it a power of 2
   n <<= 1; //making double size(2*n)
   fa.resize(n),fb.resize(n);

   fft(fa,false),fft(fb,false);
   for (size_t i=0; i<n; ++i)
      fa[i]=fa[i]*fb[i];
   fft(fa,true); //inverse fft
   r.clear();
   for (size_t i=0; i<n; ++i) {
      r.push_back(int(fa[i].r+0.5));
   }
   return ;
}

void multiplyLongNum(vector<int> &a,vector<int> &b, vector < int > &
res ) { //multiplying two long(string) numbers.(normalizing)
   reverse(a.begin(), a.end());
   reverse(b.begin(), b.end());
   multiply(a,b,res);
   int n=res.size();
   int carry = 0 ;
   for ( size_t i = 0 ; i < n ; ++ i ) {
      res [ i ] += carry ;
      carry = res [ i ] / 10 ;
      res [ i ] %= 10 ;
   }
   for(int i=(int)res.size()-1; i>0; i--) {
      if(res[i]==0) res.pop_back();
      else break;
   }
   reverse(res.begin(), res.end());
}

int main() {
   vector<int> a {1,2,9};
   vector<int> b {7,0,3,8};
   vector<int> r;
   multiplyLongNum(a,b,r);
   for(int i=0; i<r.size(); i++) {
      printf("%d",r[i]);
   }
   printf("\n");
   return 0;
}
```

96. NTT (Straight Forward)

```
#include <bits/stdc++.h>
using namespace std;

const int mod = 7340033;
const int root = 5;
const int root_inv = 4404020;  /// inverse(root, mod)
const int root_pw = 1 << 20;  /// maximum number of elements

pair<long long, long long> extended_euclid(long long a, long long b){ //
returns x, y | ax + by = gcd(a,b)
   if(b == 0)    return make_pair(1, 0);
```

```
    else{
        pair<long long, long long> d = extended_euclid(b, a%b);
        return make_pair(d.second, d.first - d.second*(a/b));
    }
}
long long inverse(long long a, long long m){
    pair<long long, long long> ret = extended_euclid(a, m);
    return ((ret.first%m)+m)%m;
}

void fft(vector<int> &a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_inv : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }

    if (invert) {
        int n_inv = inverse(n, mod);
        for (int & x : a)
            x = (int)(1LL * x * n_inv % mod);
    }
}

void multiply(vector<int> &a, vector<int> &b, vector<int> &r){
    vector<int> fa(a.begin(), a.end()),fb(b.begin(), b.end());
    size_t n = 1;
    while (n<max(a.size(),b.size())) n<<=1; //making it a power of 2
    n <<= 1; //making double size(2*n)
    fa.resize(n),fb.resize(n);
    fft(fa,false);
    fft(fb,false);
    for (size_t i=0; i<n; ++i)
        fa[i] = ( 1LL*fa[i]*fb[i] )%mod;
    fft(fa,true); //inverse fft
    r = fa;
    return;
}

int main(){
```

```
    vector<int> a = {5, 3, 2}, b = {3, 3, 4}, ans;

    multiply(a, b, ans);

    for( int i = 0; i<ans.size(); i++ ) /// resultant polynomial
        cout << i << " " << ans[i] << endl;

    return 0;
}
```

97. NTT with CRT

```
#include <bits/stdc++.h>
using namespace std;

const int mod = 1000000007;
const int p1=1012924417, p2=1004535809, p3=998244353;
const int r1=5, r2=3, r3=3;

int expmod(int k, int p, int q) {
        int res = 1;
        for ( ; p ; p >>= 1) {
                if (p & 1) res = 1ll*res*k%q;
                k = 1ll*k*k%q;
        }
        return res;
}

pair<long long, long long> extended_euclid(long long a, long long b){ //
returns x, y | ax + by = gcd(a,b)
    if(b == 0)    return make_pair(1, 0);
    else{
        pair<long long, long long> d = extended_euclid(b, a%b);
        return make_pair(d.second, d.first - d.second*(a/b));
    }
}

long long inverse(long long a, long long m){
    pair<long long, long long> ret = extended_euclid(a, m);
    return ((ret.first%m)+m)%m;
}

void fft(vector<int> &a, bool invert, int root, int p){
    int n = a.size();
    int root_inv = inverse(root, p);

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_inv : root;
        wlen = expmod(wlen, (p-1)/len, p);

        for (int i = 0; i < n; i += len) {
            int w = 1;
```

```
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % p);
                a[i+j] = u + v < p ? u + v : u + v - p;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + p;
                w = (int)(1LL * w * wlen % p);
            }
        }
    }

    if (invert) {
        int n_inv = inverse(n, p);
        for (int & x : a)
            x = (int)(1LL * x * n_inv % p);
    }
}

vector<int> u[3], v[3];

void multiply(vector<int> &a, vector<int> &b, vector<int> &r){
    vector<int> fa(a.begin(), a.end()),fb(b.begin(), b.end());
    size_t n = 1;
    while (n<max(a.size(),b.size())) n<<=1; //making it a power of 2
    n <<= 1; //making double size(2*n)
    fa.resize(n),fb.resize(n);

    for( size_t i = 0; i<3; i++ ){
        u[i].clear(), v[i].clear();
        u[i].resize(n), v[i].resize(n);
    }

    for( size_t i = 0; i<n; i++ ){
        u[0][i] = fa[i]%p1;
        u[1][i] = fa[i]%p2;
        u[2][i] = fa[i]%p3;
        v[0][i] = fb[i]%p1;
        v[1][i] = fb[i]%p2;
        v[2][i] = fb[i]%p3;
    }

    fft(u[0], false, r1, p1);
    fft(u[1], false, r2, p2);
    fft(u[2], false, r3, p3);
    fft(v[0], false, r1, p1);
    fft(v[1], false, r2, p2);
    fft(v[2], false, r3, p3);

    for (size_t i=0; i<n; ++i){
        u[0][i] = ( 1LL*u[0][i]*v[0][i] )%p1;
        u[1][i] = ( 1LL*u[1][i]*v[1][i] )%p2;
        u[2][i] = ( 1LL*u[2][i]*v[2][i] )%p3;
    }

    fft(u[0], true, r1, p1);
    fft(u[1], true, r2, p2);
    fft(u[2], true, r3, p3);

    long long s, tmp, p12b, p2b = inverse(p2, p1);
    tmp = 1LL*p1*p2%p3;
        p12b = inverse(tmp, p3);

    r.clear();
    r.resize(n);
```

```
    for( size_t i = 0; i<n; i++ ){
        tmp = ( 1LL*p2b*(u[0][i] - u[1][i])%p1 + p1 )%p1;
        s = 1LL*p2*tmp + u[1][i];
        r[i] = 1LL*p12b*((u[2][i] - s)%p3 + p3)%p3;
        r[i] = ( 1LL*r[i]*p1%mod * p2 + s)%mod;
    }

    return;
}

int main(){
//    freopen("input.txt", "r", stdin);
//    freopen("output2.txt", "w", stdout);
    int n, i, j, k, v1, v2;
    vector<int> a, b, res;

    scanf("%d", &n);
    for( i = 0; i<n; i++ ){
        scanf("%d %d", &v1, &v2);
        a.push_back(v1);
        b.push_back(v2);
    }
    multiply(a, b, res);

    for( auto it:res ){
        printf("%d\n", it);
    }

    return 0;
}
```

98. Fast Walsh Hadamard Transform

```
#define MAX (1 << 20)
#define OR 0
#define AND 1
#define XOR 2

/// Fast Walsh-Hadamard Transformation in n log n
struct fwht{
    long long P1[MAX], P2[MAX];

    void walsh_transform(long long* ar, int n, int flag = XOR){
        if (n == 0) return;

        int i, m = n/2;
        walsh_transform(ar, m, flag);
        walsh_transform(ar+m, m, flag);

        for (i = 0; i < m; i++){  /// Don't forget modulo if required
            long long x = ar[i], y = ar[i + m];
            if (flag == OR) ar[i] = x, ar[i + m] = x + y;
            if (flag == AND) ar[i] = x + y, ar[i + m] = y;
            if (flag == XOR) ar[i] = x + y, ar[i + m] = x - y;
        }
    }

    void inverse_walsh_transform(long long* ar, int n, int flag = XOR){
        if (n == 0) return;

        int i, m = n/2;
        inverse_walsh_transform(ar, m, flag);
```

```
    inverse_walsh_transform(ar+m, m, flag);

    for (i = 0; i < m; i++){  /// Don't forget modulo if required
      long long x = ar[i], y = ar[i + m];
      if (flag == OR) ar[i] = x, ar[i + m] = y - x;
      if (flag == AND) ar[i] = x - y, ar[i + m] = y;
      if (flag == XOR) ar[i] = (x + y) >> 1, ar[i + m] = (x - y) >> 1; ///
Modular inverse if required here
    }
  }

  vector <long long> convolution(int n, long long* A, long long* B, int flag
= XOR){
    assert(__builtin_popcount(n) == 1); /// n must be a power of 2
    for (int i = 0; i < n; i++) P1[i] = A[i];
    for (int i = 0; i < n; i++) P2[i] = B[i];

    walsh_transform(P1, n, flag);
    walsh_transform(P2, n, flag);
    for (int i = 0; i < n; i++) P1[i] = P1[i] * P2[i];
    inverse_walsh_transform(P1, n, flag);
    return vector<long long> (P1, P1 + n);
  }

  /// For i = 0 to n - 1, j = 0 to n - 1
  /// v[i or j] += A[i] * B[j]
  vector <long long> or_convolution(int n, long long* A, long long* B){
    return convolution(n, A, B, OR);
  }

  /// For i = 0 to n - 1, j = 0 to n - 1
  /// v[i and j] += A[i] * B[j]
  vector <long long> and_convolution(int n, long long* A, long long* B){
    return convolution(n, A, B, AND);
  }

  /// For i = 0 to n - 1, j = 0 to n - 1
  /// v[i xor j] += A[i] * B[j]
  vector <long long> xor_convolution(int n, long long* A, long long* B){
    return convolution(n, A, B, XOR);
  }
};
```

Number Theory

99. Extended Euclid (ax+by=c)

```
//ax+by=1
pair<LL,LL> egcd ( LL a, LL b ) {
   if (b == 1)
      return make_pair(0, 1);
   pair<LL,LL> ret = egcd(b%a, a);
   int p = ret.second-(b/a)*ret.first, q = ret.first;
   p %= b; //for overflow
   //cout << a << "*" << p << " + " << b << "*" << q << " = 1\n";
   return make_pair(p, -(a*p-1LL)/b);
}
//ax+by=c
bool find_any_solution( LL a, LL b, LL c, LL &x0, LL &y0, LL &g) {
   if( !a && !b ) return !c;
   g=__gcd(a,b);
   if( (c%g)!=0 )
      return false;
   a/=g;
   b/=g;
   c/=g;

   pair<LL,LL> ret=egcd(abs(a), abs(b));
   x0=ret.first;
   y0=ret.second;
   x0 = (x0*(c%b))%b;
   y0 = (c-a*x0)/b;
   if( a<0 ) x0*= -1;
   if( b<0 ) y0*= -1;
   return true;
}

void shift_solution( LL &x, LL &y, LL a, LL b, LL cnt) {
   x+= cnt*b;
   y-= cnt*a;
}
// ax+by=c;
LL find_all_solutions (LL a, LL b, LL c, LL minx, LL maxx, LL miny, LL maxy) {
//mainly takes the range
   LL x, y, g;
   if (!find_any_solution (a, b, c, x, y, g))
      return 0;
   if(!a&&!b)
      return (maxx-minx+1)*(maxy-miny+1);
   if(a&&!b) {
      x=c/a;
      if(x<minx||x>maxx) return 0;
      return maxy-miny+1;
   }
   if(!a&&b) {
      y=c/b;
      if(y<miny||y>maxy) return 0;
      return maxx-minx+1;
   }
   a /= g;
   b /= g;
   LL sign_a = a> 0?  1: - 1;
   LL sign_b = b> 0?  1: - 1;
   shift_solution (x, y, a, b, (minx - x) / b);
   if (x <minx)
```

```
    shift_solution (x, y, a, b, sign_b);
  if (x> maxx)
    return 0LL;
  LL lx1 = x;
  shift_solution (x, y, a, b, (maxx - x) / b);
  if (x> maxx)
    shift_solution (x, y, a, b, - sign_b);
  LL rx1 = x;
  shift_solution (x, y, a, b, - (miny - y) / a);
  if (y <miny)
    shift_solution (x, y, a, b, - sign_a);
  if (y> maxy)
    return 0LL;
  LL lx2 = x;
  shift_solution (x, y, a, b, - (maxy - y) / a);
  if (y> maxy)
    shift_solution (x, y, a, b, sign_a);
  LL rx2 = x;

  if (lx2> rx2)
    swap (lx2, rx2);
  LL lx = max (lx1, lx2);
  LL rx = min (rx1, rx2);

  return max(0LL,(rx - lx) / abs (b) + 1);
}
```

100. Chinese Remainder Theorem(Garner's)

```
//a=x0+x1*p0+x2*p0*p1+x3*p0*p1*p2+....+x(k-1)*p0*p1*p2**..p(k-2)
(mod p0*p1*p2*...p(k-1))
//a=remainder, r[j][i]=p[j]^-1 (mod p[i]), p=primes (0 based)
void chineseremaindertheorem(LL x[],LL a[],LL r[][100],LL p[],LL k) {
  for ( LL i = 0 ; i < k ; ++ i ) {
    x [ i ] = a [ i ] ;
    for ( LL j = 0 ; j < i ; ++ j ) {
      x [ i ] = r [ j ][ i ] * ( x [ i ] - x [ j ] ) ;
      x [ i ] = x [ i ] % p [ i ] ; //mod value to avoid overflow
      if ( x [ i ] < 0 )  x [ i ] += p [ i ] ;
    }
  }
}
```

101. Burnside Lemma

```
//LJ 1419(Necklace)
//see emaxx for theory
//Bigmod Code Need
//Sieve Code Need
#define s 1010
bool col[s];
long long prime[s]; // Prime Keep here
int relPrime(int n) { //relative prime
  int i;
  int ans=n;
  for(i=1; prime[i]*prime[i]<=n; i++)
    if(n%prime[i]==0) {
      while(n%prime[i]==0) n/=prime[i];
      ans/=prime[i];
      ans*=(prime[i]-1);
    }
  if(n>1) {
    ans/=n;
```

```
        ans*=(n-1);
    }
    return ans;
}


//most of the change were done here
LL lemmaFunction(int n,int d,int k,int m) {
    LL ans=relPrime(n);
    ans*=bigmod(k,d,m);
    ans%=m;
    return ans;
}
//burnside lemma(from emaxx)
//n and mod should be relative prime
LL burnside(int n,int k,int m) { //n=group size, k=number of color
    int i;
    LL ans=0;
    for(i=1; i*i<n; i++)
        if(n%i==0) {
            ans=(ans+lemmaFunction(n/i,i,k,m))%m;
            ans=(ans+lemmaFunction(i,n/i,k,m))%m;
        }
    if(n==i*i) ans=(ans+lemmaFunction(i,i,k,m))%m; //for ignoring double
count
    ans=(ans*bigmod(n,m-2,m))%m;
    return ans;
}
int main() {
    seive();
    int mod=1000000007;
    int t,cas=0;
    cin>>t;
    while(t--) {
        int n,k;
        scanf("%d %d",&n,&k);
        csprnt;
        print1(burnside(n,k,mod));
    }
    return 0;
}
```

102. Lucas Theorem

```
const int mod = 997;
int dp1[2007][2007];
int rec1(int nn, int rr){
    if( nn == 0 or rr == 0 )   return 1;
    if( nn==rr )    return 1;
    if( rr == 1 )   return nn%mod;
    int &ret = dp1[nn][rr];
    if( ret != -1 ) return ret;
    ret = ( 1LL*rec1(nn-1, rr) + 1LL*rec1(nn-1, rr-1) )%mod;
    return ret;
}
ll lucas(int nn, int rr){
    if( rr>nn ) return 0;
    if( nn<mod )    return rec1(nn, rr);
    ll ret = ( 1LL*lucas(nn/mod, rr/mod)*lucas(nn%mod, rr%mod) )%mod;
    return ret;
}
```

103. Inverse Module(E-GCD)

```
int extendedgcd ( int a, int b, int & x, int & y ) {
        if ( a == 0 ) {
                x = 0 ; y = 1 ;
                return b ;
        }
        int x1, y1 ;
        int d = extendedgcd( b % a, a, x1, y1 ) ;
        x = y1 - ( b / a ) * x1 ;
        y = x1 ;
        return d;
}
void findinverse(int a,int m){
  int x, y ;
  int g = extendedgcd( a, m, x, y );
  if ( g!=1 ) cout << "no solution"<<endl;
  else {
   x = ( x % m + m ) % m ;
        cout << x <<endl;
   }
}
```

104. Baby Step-Giant Step

```
//a^x=b (mod m)
int solve ( int a, int b, int m ) {
   int n = ( int ) sqrt ( m + .0 ) + 1 ;
   int an = 1 ;
   for ( int i = 0 ; i < n ; ++ i )
      an = ( an * a ) % m ;
   map < int, int > vals ;
   for ( int i = 1, cur = an ; i <= n ; ++ i ) {
      if ( ! vals. count ( cur ) )
         vals [ cur ] = i ;
      cur = ( cur * an ) % m ;
   }
   for ( int i = 0, cur = b ; i <= n ; ++ i ) {
      if ( vals. count ( cur ) ) {
         int ans = vals [ cur ] * n - i ;
         if ( ans < m )
            return ans ;
      }
      cur = ( cur * a ) % m ;
   }
   return - 1 ;
}
```

105. MillerRabin Primality Test

```
#define SZ1 10000100
#define SZ2 577145
char sieve[(SZ1>>4)+7];
int prime[SZ2];
int totP;
void bit_sieve() {
   int i,j,k,r;
   prime[0]=2;
   k=1;    totP=k;
   int lim=(int)sqrt(SZ1)+1;
   for(i=3; i<SZ1; i+=2) {
      if(!(sieve[i>>4]&(1<<((i>>1)&7)))) {
         prime[k++]=i;
         if(i<lim) {
            r=i<<1;
```

```
            for(j=i*i; j<SZ1; j+=r) {
               sieve[j>>4]|=(1<<((j>>1)&7));
            }
         }
      }
   }
   totP=k;
   return;
}
/**
1 means either n<=1
2 means prime
3 means composite square number
4 means composite non square number
*/
int miller_rabin(ll n,int it) {
   if(n<=1) return 1;
   else if(n==2) return 2;
   else {
      ll k=sqrt(n);
      for(ll i=max(0ll,k-2); i<=k+2; i++) {
         if(i*i==n) return 3;
      }
      if(n%2==0) return 4;
      else {
         ll s=0,d=n-1,a;
         while(d%2==0) {
            s++;
            d/=2;
         }
         bool f;          ll m1,m2;
            for(int i=0; i<it; i++) {
               a=prime[i];
               f=true;
               for(int j=0; j<s; j++) {
                  m1=(BigModL(a,d,n)-1+n)%n;
                  m2=(BigModL(a,(1ll<<j)*d,n)+1)%n;
                  if(m1==0||m2==0) {
                     f=false;
                     break;
                  }
               }
               if(f) return 4;
            }
         }
      }
      return 2;
}
ll get_div(ll n) {
   ll p=2,r=1,c;
   int i;
   for(i=0; i<totP && prime[i]*prime[i]<=n; i++) {
      p=prime[i];
      if(n%p==0) {
         c=1;
         while(n%p==0) {
            n/=p;
            c++;
         }
         r*=c;
      }
   }
```

```
    if(n>1) {
        if(n<=prime[totP-1]) r*=2ll;
        else r*=miller_rabin(n,12);
    }
    return r;
}
int main(void) {
    bit_sieve();    ll n;
    while(cin>>n) {
        cout<<get_div(n)<<"\n";
    }
}
```

106. Möbius function

μ(n) is defined for all positive integers n and has its values in {−1, 0, 1} depending on the factorization of n into prime factors. It is defined as follows:
μ(n) = 1 if n is a square-free positive integer with an even number of prime factors.
μ(n) = −1 if n is a square-free positive integer with an odd number of prime factors.
μ(n) = 0 if n has a squared prime factor.

```
//ray gun lightoj
int mob[MX];
int main(){
    mobius();
    scanf("%lld %lld", &a, &b);
    lli M = min(a,b);
    for(lli i = 1; i <= M; i++)
        res += mob[i]*(a/i)*(b/i); //res = 0 at first
    printf("%lld\n", res+2);
}
void mobius(void){
    for(lli i = 2; i < MX; i++) mob[i] = 4;
    mob[1] = 1;
    for(lli i = 2; i < MX; i++)
        if(mob[i] == 4){
            mob[i] = -1;
            for(lli j = i << 1; j < MX; j+=i)
            mob[j] = (mob[j] == 4)? -1:(mob[j]*(-1));
            lli ad = i*i;
            for(lli j = ad; j < MX; j += ad)
                mob[j] = 0;
        }
}
```

107. Phi Function

```
int phi[MX];
void funct(void){
    for(int i = 1; i < MX; i++)       phi[i] = i;
    for(int i = 2; i < MX; i++){
        if(phi[i] == i){
            for(int j = i ; j < MX; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

108. Find Primes (SQUFOF)

```
#define ll long long
#define fpos ffoos
```

```
ll mulmod(ll a, ll b, ll m)
{
//   __int128 xa = a, xb = b;
//   __int128 ret = xa*xb%m;
//   return (ll) ret;
   if(b==0) return 0;
   if(b&1) return (2ll * mulmod(a, b/2, m) % m + a) % m;
   return 2ll * mulmod(a, b/2, m) % m;
}
double l;
inline ll bigmod(ll a, ll p, ll m)
{
   ll ret = 1;
   while(p)
   {
      if(p&1) ret = mulmod(ret, a, m);
      a = mulmod(a, a, m);
      p/=2;
   }
   return ret;
}
inline bool millerRobin(ll p, int iter = 20)
{
   if(p==3 || p==2 || p==5) return true;
   if(p%2==0) return 0;
   if(p<3) return 0;

   for(int i = 0; i<iter; i++)
   {
      ll a = ((rand() << 15) | rand()) % (p-4) + 2;
      ll s = p - 1;
      while(s%2==0) s/=2;
      ll mod = bigmod(a, s, p);
      if(mod==1 || mod==p-1) continue;
      bool flag = 0;
      s *= 2;
      while(s != p-1)
      {
         mod = mulmod(mod, mod, p);
         if(mod==p-1){
            flag = 1;
            break;
         }
         s*=2;
      }
      if(flag==0) return 0;
   }
   return 1;
}

ll PollardRho(ll n)
{
   srand (time(NULL));
   if (n==1) return n;
   if (n % 2 == 0) return 2;
   ll x = (rand()%(n-2))+2;
   ll y = x;
   ll c = (rand()%(n-1))+1;
   ll d = 1;

   while(d==1)
   {
```

```
//      if((clock() - l)/CLOCKS_PER_SEC > 0.3) assert(0);
      x = (mulmod(x, x, n) + c)%n;
      y = (mulmod(y, y, n) + c)%n;
      y = (mulmod(y, y, n) + c)%n;
      d = __gcd(abs(x-y), n);
   }
   if(d==1 || d==n) return PollardRho(n);
   return d;
}

ll brent(ll n)
{
   if(n%2==0) return 2;
   ll y = ((rand() << 15) | rand()) % (n-1) + 1;
   ll c = ((rand() << 15) | rand()) % (n-1) + 1;
   ll m = ((rand() << 15) | rand()) % (n-1) + 1;
   ll g = 1, r = 1, q = 1, ys, x;
   while(g==1){
      x = y;
      for(int i = 0; i<r; i++)
         y = (mulmod(y, y, n) + c) % n;
      ll k = 0;
      while( k<r && g==1 ){
         ys = y;
         for(int i = 0; i<min(m, r-k); i++){
            y = (mulmod(y, y, n) + c) % n;
            q = mulmod(q , abs(x-y), n);
         }
         g = __gcd(q, n);
         k = k + m;
      }
      r *= 2;
   }
   if(g==n){
      while(true){
         ys = (mulmod(ys, ys, n) + c) % n;
         g = __gcd(abs(x-ys), n);
         if(g > 1) break;
      }
   }
   return g;
}
const int mx = 1000000;
ll prm[mx];
bitset<mx> mark;
int pos = 1;
inline void sieve()
{
   for(int i = 3; i*i<mx; i+=2)
      if(!mark[i])
         for(int j = i*i; j<mx; j+=i+i)
            mark[j] = 1;
   prm[0] = 2;
   for(int i = 3; i<mx; i+=2)
      if(!mark[i]) prm[pos++] = i;
}
ll fact[20], fpos = 0;
int cnt[20];
int dppos = 0;
pair<ll, ll> divphi[200005];
void rec(int ps, ll mult, ll phi)
{
```

```
    if(mult > 4e18 / phi) return;
    if(ps==fpos){
        if(mult==1) return;
        divphi[dppos++] = {mult, phi};
        return;
    }
    rec(ps+1, mult, phi);
    ll nw = 1;
    for(int i = 0; i<cnt[ps]; i++)
    {
        nw *= fact[ps];
        ll nwphi = phi * nw;
        nwphi -= nwphi / fact[ps];
        rec(ps+1, mult*nw, nwphi);
    }
}

const int multiplier[] = {1, 3, 5, 7, 11, 3*5, 3*7, 3*11, 5*7, 5*11, 7*11,
3*5*7, 3*5*11, 3*7*11, 5*7*11, 3*5*7*11};
#define nelems(x) (sizeof(x) / sizeof((x)[0]))
ll SQUFOF( ll N )
{
    ll D, Po, P, Pprev, Q, Qprev, q, b, r, s;
    ll L, B, i;
    s = (ll)(sqrtl(N)+0.5);
    if (s*s == N) return s;
    for (int k = 0; k < nelems(multiplier) && N <=
UINT64_MAX/multiplier[k]; k++) {
        D = multiplier[k]*N;
        Po = Pprev = P = sqrtl(D);
        Qprev = 1;
        Q = D - Po*Po;
        L = 2 * sqrtl( 2*s );
        B = 3 * L;
        for (i = 2 ; i < B ; i++) {
            b = (ll)((Po + P)/Q);
            P = b*Q - P;
            q = Q;
            Q = Qprev + b*(Pprev - P);
            r = (ll)(sqrtl(Q)+0.5);
            if (!(i & 1) && r*r == Q) break;
            Qprev = q;
            Pprev = P;
        };
        if (i >= B) continue;
        b = (ll)((Po - P)/r);
        Pprev = P = b*r + P;
        Qprev = r;
        Q = (D - Pprev*Pprev)/Qprev;
        i = 0;
        do {
            b = (ll)((Po + P)/Q);
            Pprev = P;
            P = b*Q - P;
            q = Q;
            Q = Qprev + b*(Pprev - P);
            Qprev = q;
            i++;
        } while (P != Pprev);
        r = __gcd(N, Qprev);
        if (r != 1 && r != N) return r;
    }
```

```
    return 0;
}
int main()
{
//    cout << PollardRho(21) << endl;
    srand(time(NULL));
    sieve();
//    for(int i = 0; i<25; i++)
//        cout << prm[i] << " ";
//    cout << endl;
    int t;
    scanf("%d", &t);
    while(t--)
    {
        ll fn;
        scanf("%lld", &fn);
        if(fn==1){
            printf("2\n");
            continue;
        }
        ll tmp = fn;
        fpos = 0, dppos = 0;
        for(int i = 0; i<pos; i++)
        {
            int c = 0;
            while(tmp % prm[i]==0){
                tmp /= prm[i];
                c++;
            }
            if(c) fact[fpos++] = prm[i], cnt[fpos-1] = c;
        }
        ll sq = round(sqrt(tmp));
        if(tmp==1);
        else if(sq * sq == tmp){
            fact[fpos++] = sq;
            cnt[fpos-1] = 2;
        } else if(millerRobin(tmp)){
            fact[fpos++] = tmp;
            cnt[fpos-1] = 1;
        } else{
            ll divi = SQUFOF(tmp);
            fact[fpos++] = divi;
            cnt[fpos-1] = 1;
            fact[fpos++] = tmp/divi;
            cnt[fpos-1] = 1;
        }
        rec(0, 1, 1);
        ll ans = -1;
        for(int i = 0; i<dppos; i++)
        {
            ll d = divphi[i].first, p = divphi[i].second;

            if(d > 4e18 / p) continue;
            if(divphi[i].first * divphi[i].second / 2ll == fn){
                ans = divphi[i].first;
                break;
            }
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

109. All pair GCD

```
/*uva extreme gcd
find all pair gcd
for(i=1;i<N;i++)
for(j=i+1;j<=N;j++)
   res+=gcd(i,j);   */
int phi[MX];  int sum[MX];
int main(){
   phi_function();
   sum_function();
   printf("%llu\n", sum[n]);
}
void sum_function(void){
   register int i, j;
   for(i = 2; i < MX; i++)
   {
      sum[i] += sum[i-1]+phi[i];
      for(j = i + i; j < MX; j+=i)
         sum[j] += i * phi[j/i];
   }
   return ;
}
```

110. Number Theory Notes

1.**Summation of relative Prime**=(n*phi(n))/2.

2.Summation of divisors sigma(n) = multiplication of (p^(x+1)-1)/(p-1) for all p where x is the power of p.

3.mobious function mu(n)={0, if n has one or more repeated prime (not square free) factors;
1 if n=1;
(-1)^k if n is a product of k distinct primes;}

Counted using seive with initialize all with 1.

4. **Lucas Theorem**: Find C(n,k)%p where p is prime and n and k are converted into base p numbers and now inidividually multiplying the digit combination.

5. **A ^ x = A ^ (x% Phi (C) + Phi (C)) (mod C) (X>=Phi(C))**

*****All division is integer division**

nCr(n,r) = nCr(n-1,r)+ nCr(n-1,r-1);

n = p1e1*p2e2*p3e3*.....

Euler's totient phi of n = number of integers k gcd(k,n) == 1(1 to n)

Phi(n) = n(1-1/p1)(1-1/p2)(1-1/p3).....

= p1(e1-1)(p1-1)*p2(e2-1)(p2-1)*p3(e3-1)(p3-1)*.....

= phi[n] – (phi[n]/p) ///p = 2 to n, p is prime factor of n

Sigma function

sigma zero = number of divisor

= (e1+1)(e2+1)(e3+1).....

sigma one = sum of divisor

= multiple of all p^(e+1)-1/p-1 p

///p = 2 to n, p is prime factor of n

sigma x = sum of dx

///d is the divisor of n (1 to n)

= multiple of all p(e+1)*x-1/px-1

///p = 2 to n, p is prime factor of n

Catalan numbers = $1/(n+1)*(^{2n}C_n)$

= $(^{2n}C_n)$- $(^{2n}C_{n+1})$

= $C_0 = 1$ and C_{n+1} = sum of C_iC_{n-i} (I = 0 to n)

=multiple of (n+k)/k (k = 2 to n) (not integer division)

= 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786

Application:

1.Cn is the number of Dyck words[2] of length 2n. A Dyck word is a string consisting of n X's and n Y's such that no initial segment of the string has

more Y's than X's (see also Dyck language). For example, the following are the Dyck words of length 6:
XXXYYY XYXXYY XYXYXY XXYYXY XXYXYY.
2.Re-interpreting the symbol X as an open parenthesis and Y as a close parenthesis, Cn counts the number of expressions containing n pairs of parentheses which are correctly matched:
((())) ()(()) ()()() (())() (()())
3.Cn is the number of different ways n + 1 factors can be completely parenthesized (or the number of ways of associating n applications of a binary operator). For n = 3, for example, we have the following five different parenthesizations of four factors:
((ab)c)d (a(bc))d (ab)(cd) a((bc)d) a(b(cd))
4.The associahedron of order 4 with the C4=14 full binary trees with 5 leaves
Successive applications of a binary operator can be represented in terms of a full binary tree. (A rooted binary tree is full if every vertex has either two children or no children.) It follows that Cn is the number of full binary trees with n + 1 leaves:
5. Cn is the number of different ways a convex polygon with n + 2 sides can be cut into triangles by connecting vertices with straight lines (a form of Polygon triangulation)
6. Cn is the number of ways to tile a stairstep shape of height n with n rectangles

Miscellaneous

111. Big Integer

```
struct Bigint {
   string a; // to store the digits
   int sign; // sign = -1 for negative numbers, sign = 1 otherwise

   Bigint() {} // default constructor
   Bigint( string b ) {
      (*this) = b;    // constructor for string
   }
   Bigint( long long num ) {
      if(num<0) sign=-1;
      else sign=1;
      if(num==0)  a.push_back('0');
      while(num) {
         a.push_back( num%10 + '0');
         num/=10;
      }
   } // constructor for string

   int size() { // returns number of digits
      return a.size();
   }
   Bigint inverseSign() { // changes the sign
      sign *= -1;
      return (*this);
   }

   Bigint normalize( int newSign ) { // removes leading 0, fixes sign
      for( int i = a.size() - 1; i > 0 && a[i] == '0'; i-- )
         a.erase(a.begin() + i);
      sign = ( a.size() == 1 && a[0] == '0' ) ? 1 : newSign;
      return (*this);
   }

   void operator = ( string b ) { // assigns a string to Bigint
      a = b[0] == '-' ? b.substr(1) : b;
      reverse( a.begin(), a.end() );
      this->normalize( b[0] == '-' ? -1 : 1 );
   }

   bool operator < ( const Bigint &b ) const { // less than operator
      if( sign != b.sign ) return sign < b.sign;
      if( a.size() != b.a.size() )
         return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size();
      for( int i = a.size() - 1; i >= 0; i-- ) if( a[i] != b.a[i] )
            return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
      return false;
   }
   bool operator == ( const Bigint &b ) const {
// operator for equality
      return a == b.a && sign == b.sign;
   }
   Bigint operator + ( Bigint b ) { // addition operator overloading
      if( sign != b.sign ) return (*this) - b.inverseSign();
      Bigint c;
      for(int i = 0, carry = 0; i<a.size() || i<b.size() || carry; i++ ) {
         carry+=(i<a.size() ? a[i]-48 : 0)+(i<b.a.size() ? b.a[i]-48 : 0);
         c.a += (carry % 10 + 48);
         carry /= 10;
```

```
    }
    return c.normalize(sign);
}
Bigint operator - ( Bigint b ) { // subtraction operator overloading
    if( sign != b.sign ) return (*this) + b.inverseSign();
    int s = sign;
    sign = b.sign = 1;
    if( (*this) < b ) return ((b - (*this)).inverseSign()).normalize(-s);
    Bigint c;
    for( int i = 0, borrow = 0; i < a.size(); i++ ) {
        borrow = a[i] - borrow - (i < b.size() ? b.a[i] : 48);
        c.a += borrow >= 0 ? borrow + 48 : borrow + 58;
        borrow = borrow >= 0 ? 0 : 1;
    }
    return c.normalize(s);
}
Bigint operator * ( Bigint b ) {
  // multiplication operator overloading
    int MAXN=a.size()+b.size()+5;
    int tmp[MAXN];
    memset(tmp,0,sizeof(tmp));
    for(int i=0; i<a.size(); i++)
        for(int j=0, p=i; j<b.size(); j++) {
            tmp[p++] += (a[i]-'0')*(b.a[j]-'0');
        }
    Bigint c;
    for(int i=0; i<MAXN-1; i++) {
        tmp[i+1] += tmp[i]/10;
        tmp[i] %= 10;
        c.a.push_back(tmp[i]+'0');
    }
    return c.normalize(sign*b.sign);

}

Bigint operator / ( Bigint b ) { // division operator overloading
    if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
    Bigint c("0"), d;
    for( int j = 0; j < a.size(); j++ ) d.a += "0";
    int dSign = sign * b.sign;
    b.sign = 1;
    for( int i = a.size() - 1; i >= 0; i-- ) {
        c.a.insert( c.a.begin(), '0');
        c = c + a.substr( i, 1 );
        while( !( c < b ) ) c = c - b, d.a[i]++;
    }
    return d.normalize(dSign);
}
Bigint operator % ( Bigint b ) { // modulo operator overloading
    if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
    Bigint c("0");
    b.sign = 1;
    for( int i = a.size() - 1; i >= 0; i-- ) {
        c.a.insert( c.a.begin(), '0');
        c = c + a.substr( i, 1 );
        while( !( c < b ) ) c = c - b;
    }
    return c.normalize(sign);
}
void print() {
    if( sign == -1 ) putchar('-');
    for( int i = a.size() - 1; i >= 0; i-- ) putchar(a[i]);
```

```
      puts("");
   }
};
```

112. Ternary Search

```
/// Ternary Search for finding the Point with minimum Value

double low = 0, hi = (double)INF;
for(int i = 0; i < 100; i++){
   double a = (hi+2.0*low)/3.0;
   double b = (2.0*hi+low)/3.0;
   //deb(low,hi,a,b);
   if( solve(a) < solve(b) ){
      low = a;
   }
   else{
      hi = b;
   }
}
printf("%.5lf\n", solve((low+hi)/2.0);

/// End of Ternary Search

/// Ternary Search on Integer values

int lo = -1, hi = n;
while( hi-lo>1 ){
   int mid = (hi+lo)>>1;
   if( f(mid)>f(mid+1) )
      hi = mid;
   else
      lo = mid;
}
/// (lo+1) is the answer
```

113. Stable Marriage Problem

```
// Number of Men or Women
// O based
#define lim  150
int prefer[2*lim][lim]; //preference for woman and man

// This function returns true if woman 'w' prefers man 'm1' over man 'm'
bool wPrefersM1OverM(int N, int w, int m, int m1) {
   // Check if w prefers m over her current engagment m1
   for (int i = 0; i < N; i++) {
      // If m1 comes before m in list of w, then w prefers her
      // current engagement, don't do anything
      if (prefer[w][i] == m1)
         return true;

      // If m cmes before m1 in w's list, then free her current
      // engagement and engage her with m
      if (prefer[w][i] == m)
         return false;
   }
}

// Prints stable matching for N boys and N girls. Boys are numbered as 0
to
// N-1. Girls are numbereed as N to 2N-1.
void stableMarriage(int N) {
   // Stores partner of women. This is our output array that
```

```
// stores paing information.  The value of wPartner[i]
// indicates the partner assigned to woman N+i.  Note that
// the woman numbers between N and 2*N-1. The value -1
// indicates that (N+i)'th woman is free
int wPartner[N];

// An array to store availability of men.  If mFree[i] is
// false, then man 'i' is free, otherwise engaged.
bool mFree[N];

// Initialize all men and women as free
memset(wPartner, -1, sizeof(wPartner));
memset(mFree, false, sizeof(mFree));
int freeCount = N;

// While there are free men
while (freeCount > 0) {
   // Pick the first free man (we could pick any)
   int m;
   for (m = 0; m < N; m++)
      if (mFree[m] == false)
         break;

   // One by one go to all women according to m's preferences.
   // Here m is the picked free man
   for (int i = 0; i < N && mFree[m] == false; i++) {
      int w = prefer[m][i];
      // The woman of preference is free, w and m become
      // partners (Note that the partnership maybe changed
      // later). So we can say they are engaged not married
      if (wPartner[w-N] == -1) {
         wPartner[w-N] = m;
         mFree[m] = true;
         freeCount--;
      }
      else { // If w is not free
         // Find current engagement of w
         int m1 = wPartner[w-N];
         // If w prefers m over her current engagement m1,
         // then break the engagement between w and m1 and
         // engage m with w.
         if (wPrefersM1OverM(N, w, m, m1) == false) {
            wPartner[w-N] = m;
            mFree[m] = true;
            mFree[m1] = false;
         }
      } // End of Else
   } // End of the for loop that goes to all women in m's list
} // End of main while loop

   // Print the solution
   for (int i = 0; i < N; i++)
      printf(" (%d %d)",wPartner[i]+1,i+1+N);
   printf("\n");
}
// Driver program to test above functions
int main() {
   int t,cas=0;
   cin>>t;
   while(t--) {
      int n;
      cin>>n;
```

```
    int i,j;
    for(i=0; i<2*n; i++) {
      for(j=0; j<n; j++) {
        cin>>prefer[i][j];
        prefer[i][j]--;
      }
    }
    printf("Case %d:",++cas);
    stableMarriage(n);
  }
  return 0;
}
/*
Sample Input
1
3
4 5 6
6 5 4
5 4 6
2 1 3
1 2 3
3 2 1
Sample Input
Case 1: (2 6) (1 4) (3 5)
*/
```

114. 3D LIS

```
//complexity n(logn)^2
const int MAXN = 300110;
struct node {
  int x,y,z;
} box[300111];
map <int, int> pos[MAXN];
map <int, int>::iterator it;
int m, n, A, B;
int C = ~(1<<31), M = (1<<16)-1;
int r() {
  A = 36969 * (A & M) + (A >> 16);
  B = 18000 * (B & M) + (B >> 16);
  return (C & ((A << 16) + B)) % 1000000;
}
int cmp(const node & a, const node & b) {
  if(a.x != b.x) return a.x < b.x;
  if(a.y != b.y) return a.y > b.y;
  return 0;
}
bool check(int a, int b) {
  if(pos[a].empty()) return false;
  it = pos[a].lower_bound(box[b].y);
  if(it != pos[a].begin()) {
    it--;
    if(it->second < box[b].z) return true;
  }
  return false;
}
//y should be strictly increasing, and z should be strictly decreasing
void insert(int a, int b) {
  if(pos[a].empty()) {
    pos[a][box[b].y] = box[b].z;
    return ;
  }
  it = pos[a].lower_bound(box[b].y);
```

```
    if(it == pos[a].end()) {
        it--;
        if(it->second <= box[b].z) return ;
        pos[a][box[b].y] = box[b].z;
        return ;
    }
    if(it->first == box[b].y) {
        if(it->second <= box[b].z) {
            return ;
        }
    }
    if(it != pos[a].begin()) {
        if((--it)->second <= box[b].z) return ;
        it++;
    }
    while(it != pos[a].end() && it->second >= box[b].z) {
        pos[a].erase(it++);
    }
    pos[a][box[b].y] = box[b].z;
}

int main() {
    //freopen("pro.in", "r", stdin);
    while(scanf("%d%d%d%d", &m, &n, &A, &B)) {
        if(m == 0 && n == 0 && A == 0 && B == 0) break;
        for(int i = 1; i <= m; i++) {
            scanf("%d%d%d", &box[i].x, &box[i].y, &box[i].z);
        }
        for(int i = 0; i < MAXN; i++) pos[i].clear();
        for(int i = 1; i <= n; i++) {
            box[i + m].x = r();
            box[i + m].y = r();
            box[i + m].z = r();
        }
        n += m;
        int f_ans = 1;
        sort(box + 1, box + 1 + n, cmp);
        int mx = 0;
        for(int i = 1; i <= n; i++) {
            if(i > 1 && box[i].x == box[i - 1].x &&
              box[i].y == box[i - 1].y && box[i].z == box[i - 1].z) continue;
            int l = 1, r = mx, mid, ans = 0;
            while(l <= r) {
                mid = (l + r) / 2;
                if(check(mid, i)) {
                    l = mid + 1;
                    ans = mid;
                } else {
                    r = mid - 1;
                }
            }
            f_ans = max(f_ans, ans + 1);
            insert(ans + 1, i);
            mx = f_ans;
        }
        printf("%d\n", f_ans);
    }
}
```

115. Dates

```
string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
// converts Gregorian date to integer (Julian day number)
```

```
int dateToInt (int m, int d, int y) {
   return
      1461 * (y + 4800 + (m - 14) / 12) / 4 +
      367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
      3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
      d - 32075;
}
// converts integer (Julian day number) to Gregorian date:
month/day/year
void intToDate (int jd, int &m, int &d, int &y) {
   int x, n, i, j;
   x = jd + 68569;
   n = 4 * x / 146097;
   x -= (146097 * n + 3) / 4;
   i = (4000 * (x + 1)) / 1461001;
   x -= 1461 * i / 4 - 31;
   j = 80 * x / 2447;
   d = x - 2447 * j / 80;
   x = j / 11;
   m = j + 2 - 12 * x;
   y = 100 * (n - 49) + i + x;
}
// converts integer (Julian day number) to day of week
string intToDay (int jd) {
   return dayOfWeek[jd % 7];
}
int main (int argc, char **argv) {
   int jd = dateToInt (3, 24, 2004);
   int m, d, y;
   intToDate (jd, m, d, y);
   string day = intToDay (jd);
   // expected output:
   //   2453089
   //   3/24/2004
   //   Wed
   cout << jd << endl
      << m << "/" << d << "/" << y << endl
      << day << endl;
}
```

116. Latitude Longitude

```
/*Converts from rectangular coordinates to latitude/longitude and vice
versa. Uses degrees (not radians). */
struct ll {
   double r, lat, lon;
};
struct rect {
   double x, y, z;
};
ll convert(rect& P) {
   ll Q;
   Q.r = sqrt(P.x*P.x+P.y*P.y+P.z*P.z);
   Q.lat = 180/M_PI*asin(P.z/Q.r);
   Q.lon = 180/M_PI*acos(P.x/sqrt(P.x*P.x+P.y*P.y));

   return Q;
}
rect convert(ll& Q) {
   rect P;
   P.x = Q.r*cos(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
   P.y = Q.r*sin(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
   P.z = Q.r*sin(Q.lat*M_PI/180);
```

```
    return P;
}
int main() {
    rect A;
    ll B;
    A.x = -1.0;
    A.y = 2.0;
    A.z = -3.0;
    B = convert(A);
    cout << B.r << " " << B.lat << " " << B.lon << endl;
    A = convert(B);
    cout << A.x << " " << A.y << " " << A.z << endl;
}
```

117. Knights Move in infinity grid

```
ll distance(ll sx, ll sy, ll tx, ll ty) {
    ll x, y, t;
    double delta;
    // special corner cases
    if (test(1, 1, 2, 2) ||
        test(7, 7, 8, 8) ||
        test(7, 2, 8, 1) ||
        test(1, 8, 2, 7))
        return 4;
    // axes symmetry
    x = abs(sx - tx);
    y = abs(sy - ty);
    // diagonal symmetry
    if (x < y) {
        t = x;
        x = y;
        y = t;
    }
    // 2 corner cases
    if (x == 1 && y == 0)
        return 3;
    if (x == 2 && y == 2)
        return 4;
    // main
    delta = x - y;
    if (y > delta) {
        return (ll)(delta - 2 * floor((delta - y) / 3));
    }
    else {
        return (ll)(delta - 2 * floor((delta - y) / 4));
    }
}
```

118. Infix to Postfix

```
int prec[300];//precidence (it should be filled by user)
//make postfix notation with variables and numbers with proper
bracketing
void postfix(string &a,string &b) {
    b.clear();
    a.pb(')');
    stack<char>s;
    s.push('(');
    char tem;
    int i;
    for(i=0; i<SZ(a); i++) {
        if(a[i]==')') { //closing bracket
            while(s.size()) {
```

```
            tem=s.top();
            s.pop();
            if(tem=='(') break;
            b.push_back(' ');
            b.push_back(tem);
            b.push_back(' ');
        }
    }
    else if(prec[a[i]]) { //operators
        b.pb(' ');
        while(s.size()) {
            tem=s.top();
            if(prec[tem]<prec[a[i]]) break;
            s.pop();
            b.push_back(' ');
            b.push_back(tem);
            b.push_back(' ');
        }
        s.push(a[i]);
    }
    else if(a[i]=='(') {
        b.push_back(' ');    //opening bracket
        s.push(a[i]);
    }
    else if(isalpha(a[i])) { //variable (size 1)
        b.push_back(' ');
        b.push_back(a[i]);
        b.push_back(' ');
    }
    else b.push_back(a[i]); //number
}
}
/*
3*2*(2*x+2)+2*x*(3+2-1)=2*x
3*2*(2)*(x)+2*x*(3+2)=x*2*(2)+2*((x)*(2+3))+2+((2)+(3))
*/
```

119. SStream

```
string val;
stringstream ss (stringstream::in | stringstream::out);
ss << "120 ab 377 6 5 2000";
while(ss>>val)
{
  //ss >> val;
  cout << val << endl;
}
```

120. Maximum Disjoint Segment In an Interval

```
#define lson node*2,beg,mid
#define rson node*2+1,mid+1,end
int L[500005], points[500005],
    min_tree[300005 * 8], max_tree[300005 * 8],
    lazy1[300005 * 8], lazy2[300005 * 8],
    ans[500005];
struct line_data {
    int x, y;
};
line_data line[100005];
struct qryy_data {
    int l, r, id;
};
qryy_data qry[100006];
```

```
bool comp(qryy_data a, qryy_data b) {
   return a.r < b.r;
}
void refresh(int node, int beg, int end) {
   if(lazy1[node]) {
      if(beg != end) {
         min_tree[node * 2] = max_tree[node * 2] = min_tree[node * 2 + 1]
= max_tree[node * 2 + 1] = lazy1[node];
         lazy1[node * 2] = lazy1[node * 2 + 1]  = lazy1[node];
      }
      lazy1[node] = 0;
   }
   if(lazy2[node]) {
      if(beg != end) {
         lazy2[node * 2] += lazy2[node];
         lazy2[node * 2 + 1] += lazy2[node];
         lazy2[node] = 0;
      }
   }
}
void build(int node, int beg, int end) {
   lazy1[node] = lazy2[node] = 0;
   if(beg == end) {
      min_tree[node] = beg;
      max_tree[node] = beg;
      return;
   }
   int mid = (beg + end) / 2;
   build(lson);
   build(rson);
   min_tree[node] = min(min_tree[node * 2], min_tree[node * 2 + 1]);
   max_tree[node] = max(max_tree[node * 2], max_tree[node * 2 + 1]);
}

void update(int node, int beg, int end, int i, int j, int c, int d) {
   refresh(node, beg, end);
   if(beg > j || end < i) return;
   if(beg >= i && end <= j) {
      if(c >= max_tree[node]) {
         lazy2[node]++;
         max_tree[node] = min_tree[node] = d;
         lazy1[node] = d;
      } else if(c < min_tree[node]) return ;
      else {
         int mid = (beg + end) / 2;
         update(lson, i, j, c, d);
         update(rson, i, j, c, d);
         min_tree[node] = min(min_tree[node * 2], min_tree[node * 2 +
1]);
         max_tree[node] = max(max_tree[node * 2], max_tree[node * 2 +
1]);
      }
      return ;
   }
   int mid = (beg + end) / 2;
   update(lson, i, j, c, d);
   update(rson, i, j, c, d);
   min_tree[node] = min(min_tree[node * 2], min_tree[node * 2 + 1]);
   max_tree[node] = max(max_tree[node * 2], max_tree[node * 2 + 1]);
}

int query(int node, int beg, int end, int i) {
```

```
    if(beg >= i && end <= i) return lazy2[node];
    refresh(node, beg, end);
    int mid = (beg + end) / 2;
    if(i <= mid) return query(lson, i);
    else return query(rson, i);
}
int main() {
    int n, m;
    while(sf2(n, m) == 2) {
        int cnt = 0;
        for(int i = 1; i <= n; i++) {
            sf2(line[i].x, line[i].y);
            points[++cnt] = line[i].x;
            points[++cnt] = line[i].y;
        }
        for(int i = 1; i <= m; i++) {
            sf2(qry[i].l, qry[i].r);
            qry[i].id = i;
            points[++cnt] = qry[i].l;
            points[++cnt] = qry[i].r;
        }
        sort(points + 1, points + 1 + cnt);
        cnt = unique(points + 1, points + 1 + cnt) - points - 1;
        mem(L, -1);
        clr(ans);
        for(int i = 1; i <= n; i++) {
            line[i].x = lower_bound(points + 1, points + cnt + 1, line[i].x) -
points;
            line[i].y = lower_bound(points + 1, points + cnt + 1, line[i].y) -
points;
            L[ line[i].y ] = max(L[ line[i].y ], line[i].x);
        }
        for(int i = 1; i <= m; i++) {
            qry[i].l = lower_bound(points + 1, points + 1 + cnt, qry[i].l) - points;
            qry[i].r = lower_bound(points + 1, points + 1 + cnt, qry[i].r) - points;
        }
        build(1, 1, cnt);
        int p = 1;
        sort(qry + 1, qry + 1 + m, comp);
        for(int i = 1; i <= cnt; i++) {
            if(L[i] != -1) {
                update(1, 1, cnt, 1, i, L[i], i);
            }
            while(qry[p].r == i) {
                ans[qry[p].id] = query(1, 1, cnt, qry[p].l);
                p++;
            }
        }
        for(int i = 1; i <= m; i++) {
            pf("%d\n", ans[i]);
        }
    }
}
```

Geometry

121. Line Intersection Integer

```
typedef long long ll;
typedef struct {
   ll x,y;
   void scan() {
      cin>>x>>y;
   }
} P;

P MV(P a,P b) {
   P r;
   r.x = b.x-a.x;
   r.y = b.y-a.y;
   return r;
}
ll CV(P a,P b) {
   return a.x*b.y - a.y*b.x;
}

bool onsegment(P a,P b,P c) {
   return ( min(a.x,b.x)<=c.x && c.x<=max(a.x,b.x) && min(a.y,b.y)<=c.y
&& c.y<=max(a.y,b.y) ) ;
}
bool segment_intersect(P p1,P p2,P p3,P p4) {
   ll d1,d2,d3,d4;

   d1 = CV(MV(p3,p4),MV(p3,p1));
   d2 = CV(MV(p3,p4),MV(p3,p2));
   d3 = CV(MV(p1,p2),MV(p1,p3));
   d4 = CV(MV(p1,p2),MV(p1,p4));

   if(d1*d2<0 && d3*d4<0)  return true;
   if(!d1 && onsegment(p3,p4,p1))  return true;
   if(!d2 && onsegment(p3,p4,p2))  return true;
   if(!d3 && onsegment(p1,p2,p3))  return true;
   if(!d4 && onsegment(p1,p2,p4))  return true;

   return false;
}
```

122. Bikpik Colonies

```
#define EPS 1e-9
#define pi acos(-1.0)
int cmp(double x)
{
   if(fabs(x) < EPS) return 0;
   return x < 0 ? -1 : 1;
}
struct PT
{
   double x, y;
   PT()
   {
      x = y = 0;
   }
   PT(double _x, double _y)
   {
      x = _x, y = _y;
   }
```

```
PT operator-(const PT &a) const
{
   return PT(x - a.x, y - a.y);
}
PT operator+(const PT &a) const
{
   return PT(x + a.x, y + a.y);
}
PT operator*(double a) const
{
   return PT(x * a, y * a);
}
PT operator/(double a) const
{
   return PT(x / a, y / a);
}
bool operator<(PT p) const
{
   return x<p.x || (x==p.x && y<p.y);
}
double val()
{
   return sqrt(x * x + y * y);
}
void scan()
{
   scanf("%lf %lf", &x, &y);
}
void print()
{
   printf("(%.4f, %.4f)", x, y);
   }
};
struct line
{
   double a, b, c;
};

double dist(PT a, PT b)
{
   return (a - b).val();
}

double dist2(PT a, PT b)
{
   a = a - b;
   return a.x * a.x + a.y * a.y;
}
double dot(PT a, PT b)
{
   return a.x * b.x + a.y * b.y;
}
double cross(PT a, PT b)
{
   return a.x * b.y - a.y * b.x;
}
PT RotateCCW90(PT p)
{
   return PT(-p.y,p.x);
}
PT RotateCW90(PT p)
{
```

```
    return PT(p.y,-p.x);
}
PT RotateCCW(PT p, double t)
{
    return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}
PT RotateCW(PT p, double t)
{
    return PT(p.x*cos(t)+p.y*sin(t), -p.x*sin(t)+p.y*cos(t));
}

// project PT c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c)
{
    double r = dot(b-a,b-a);
    if (fabs(r) < EPS) return a;
    r = dot(c-a, b-a)/r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c)
{
    return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}
// returns bisector of angle YXZ
line bisector(PT Y, PT X, PT Z)
{
    PT xy = (Y - X)/(Y - X).val();
    PT xz = (Z - X)/(Z - X).val();
    PT d = xy + xz;
    line ret{d.y, -d.x, d.x * X.y - d.y * X.x};
    return ret;
}

struct point3
{
    double x, y, z;
    point3() {}
    point3(double _x, double _y, double _z)
    {
        x = _x, y = _y, z = _z;
    }
    point3 operator-(const point3 &a) const
    {
        return point3(x - a.x, y - a.y, z - a.z);
    }
    point3 operator+(const point3 &a) const
    {
        return point3(x + a.x, y + a.y, z + a.z);
    }
    point3 operator*(double a) const
    {
        return point3(x * a, y * a, z*a);
    }
    point3 operator/(double a) const
    {
        return point3(x / a, y / a, z/a);
    }
    double val()
```

```
    {
        return sqrt(x*x + y*y + z*z);
    }
    void scan()
    {
        scanf("%lf %lf %lf", &x, &y, &z);
    }
};


inline vector<PT> convex_hull(vector<PT> P)
{
    int n = P.size(), k = 0;
    vector<PT> H(2*n);
    sort(P.begin(), P.end());
    for(int i = 0; i<n; i++)
    {
        while(k>=2 && cmp(cross(H[k-2] - H[k-1], P[i] - H[k-1])) <= 0) k--;
        H[k++] = P[i];
    }
    for(int i = n-2, t = k+1; i>=0; i--)
    {
        while(k>=t && cmp(cross(H[k-2] - H[k-1], P[i] - H[k-1])) <= 0) k--;
        H[k++] = P[i];
    }
    H.resize(k-1);
    return H;
}


point3 cross3(point3 a, point3 b)
{
    return point3(a.y*b.z - a.z*b.y, a.z*b.x-a.x*b.z, a.x*b.y-a.y*b.x);
}

point3 rotatex(point3 a, double angle)
{
    double cosa = cos(angle), sina = sin(angle);
    double y = a.y * cosa - a.z * sina;
    double z = a.y * sina + a.z * cosa;
    double x = a.x;
    return point3(x, y, z);
}

point3 rotatey(point3 a, double angle)
{
    double cosa = cos(angle), sina = sin(angle);
    double z = a.z * cosa - a.x * sina;
    double x = a.z * sina + a.x * cosa;
    double y = a.y;
    return point3(x, y, z);
}

point3 rotatez(point3 a, double angle) // guess perfect
{
    double cosa = cos(angle), sina = sin(angle);
    double x = a.x * cosa - a.y * sina;
    double y = a.x * sina + a.y * cosa;
    double z = a.z;
    return point3(x, y, z);
}
```

```
double d;
double solve(vector<PT> vt)
{
   double peri = 0.0, area = 0.0;
   auto hull = convex_hull(vt);
   int n = hull.size();
   for(int i = 0; i<n; i++)
   {
      int a = i;
      int b = (i + 1) % n;
      peri += dist(hull[a], hull[b]);
      area += hull[a].x * hull[b].y - hull[a].y * hull[b].x;
   }
   area = fabs(area/2.0);
   return area * 2.0+ pi * d * peri + 4 * pi * d * d;
}

vector<PT> vt;
int n;
double dp2[(1<<13)];
int vis2[(1<<13)],cas;
double cal(int mask)
{
   if(mask==0)return 0;
   if(vis2[mask]==cas)return dp2[mask];
   vis2[mask]=cas;
   vector<PT> nw;
   for(int i = 0; i<n; i++)
   {
      if(mask & (1<<i)) nw.push_back(vt[i]);
   }
   return dp2[mask]=solve(nw);
}
const double big = 1000000000000000000.0;
double dp3[1<<13];
double dp_func2(int n)
{

   dp3[0]=0;
   for(int i=1; i<(1<<n); i++)
   {
      dp3[i]=cal(i);
      double dim=dp3[i],tp;

      for(int j=i;; j=(j-1)&i)
      {
         if(i==j) continue;
         if(j==0)break;
         tp=dp3[j]+dp3[i^j];
         dim=min(dim,tp);
      }
      // cout<<i<<endl;
      dp3[i]=dim;
   }

   return dp3[(1<<n)-1];

}

double dot3(point3 a, point3 b)
{
   return a.x * b.x + a.y * b.y + a.z * b.z;
```

```
}
int main()
{
//   freopen("input.txt", "r", stdin);
//   freopen("output.txt", "w", stdout);
   int t, tst = 1;
   cas = 0;
   scanf("%d", &t);
   while(t--)
   {
      cas++;
      scanf("%d %lf", &n, &d);
//       assert(n>0);
      vector<point3> ara = vector<point3>(n);
      point3 norm(0, 1, 0);
      for(int i = 0; i<n; i++)
         ara[i].scan();
      vt = vector<PT>(n);
      int f = 0;
      for(int i = 2; i<n; i++)
      {
         if(cmp(cross3(ara[1] - ara[0], ara[i] - ara[0]).val()) != 0)
         {
            norm = cross3(ara[1] - ara[0], ara[i] - ara[0]);
            norm = norm / norm.val();
            f = 1;
            break;
         }
      }
      if(!f)
      {
         if(n>1)
         {
            norm = ara[1] - ara[0];
            if(cmp(norm.x) != 0)
            {
               norm = point3(-(norm.y + norm.z) / norm.x, 1, 1);
            }
            else if(cmp(norm.y) != 0)
            {
               norm = point3(1, -(norm.x + norm.z) / norm.y, 1);
            }
            else if(cmp(norm.z) != 0)
            {
               norm = point3(1, 1, -(norm.y + norm.x) / norm.z);
            }
//                assert(cmp(norm.val()));
            norm = norm / norm.val();
         }
      }
      cout<<setprecision(20);
      point3 prvNorm = norm;
      double anglez = atan2(norm.y, norm.z);
//       cout << norm.x << " - " << norm.y << " - " << norm.z << endl;
//       cout << anglez  << " ***" << endl;
      norm = rotatex(norm, anglez);
//       cout << norm.x << " - " << norm.y << " - " << norm.z << endl;
      double anglez2 = -atan2(norm.x, norm.z);
      norm = rotatey(norm, anglez2);
//       cout << norm.x << " - " << norm.y << " - " << norm.z << endl;
//       cout << acos(norm.z)  << " ***" << endl;
//       assert(cmp(fabs(norm.z) - 1)==0);
```

```
//      assert(cmp(norm.x)==0);
//      assert(cmp(norm.y)==0);
      double allz = 0;
      point3 xx = ara[0];
      for(int i = 0; i<n; i++)
      {
         assert(cmp(dot3(ara[i] - xx, prvNorm)) == 0);
         ara[i] = rotatex(ara[i], anglez);
         ara[i] = rotatey(ara[i], anglez2);
//             printf("%.3f %.3f %.3f\n", ara[i].x, ara[i].y, ara[i].z);
         vt[i] = PT(ara[i].x, ara[i].y);
         if(i==0) allz = ara[i].z;
         else
         {
//             assert(cmp(fabs(allz) - fabs(ara[i].z)) == 0);
         }
      }
      double ans = dp_func2(n);
      printf("%.14f\n", ans);
   }
   return 0;
}
```

123. Closest Pair of Point

```
typedef pair<int,int>pii;
struct P {
   double x,y,z;
   P(double xt=0,double yt=0,int zt=0) {
      x=xt,y=yt,z=zt;
   }
};
struct Comparator {
   bool operator ()(const P &a,const P &b)
   const {
      if(a.y!=b.y) return a.y<b.y;
      return a.x<b.x;
   }
};
const int S = 100000;
P p[S];
bool com(P a,P b) {
   return(a.x!=b.x)?(a.x<b.x):(a.y<b.y);
}
double SD(P a,P b) {
   return sqr(a.x-b.x)+sqr(a.y-b.y);
}
pii ClosestPair(P p[],int n) {
/// Return the index's of closest points.
   int left,right,ci,cj,i;
   double dis,m;
   set<P,Comparator>st;
   P tmp;
   __typeof(st.begin()) itl,ith;
   sort(p,p+n,com);
   for(i=0; i<n; i++) p[i].z = i;
   ci=p[0].z;
   cj=p[1].z;
   m = SD(p[0],p[1]);
   st.insert(p[0]);
   st.insert(p[1]);
   left=0;
   right=2;
```

```
  while(right<n) {
    while(left<right&&sqr(p[left].x-p[right].x)>=m) {
      st.erase(p[left]);
      left++;
    }
    dis=sqrt(m)+ERR;
    itl = st.lower_bound(P(p[right].x,
                  p[right].y-dis));
    ith = st.upper_bound(P(p[right].x,
                  p[right].y+dis));
    while(itl!=ith) {
      dis = SD(*itl,p[right]);
      if(dis<m) {
        m=dis;
        ci=itl->z;
        cj = p[right].z;
      }
      itl++;
    }
    st.insert(p[right]);
    right++;
  }
  return pii(ci,cj);
}
```

124. Geometry 2D (Rumman Bhai)

```
#define PI acos(-1.0)
using namespace std;
const double INF = 1e100;
const double EPS = 1e-9;
int EQ(double x)
{
  if(fabs(x)<EPS) return 0;
  else if(x>0) return 1;
  else return -1;
}
struct PT
{
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const
  {
    return PT(x+p.x, y+p.y);
  }
  PT operator - (const PT &p)  const
  {
    return PT(x-p.x, y-p.y);
  }
  PT operator * (double c)     const
  {
    return PT(x*c,   y*c  );
  }
  PT operator / (double c)     const
  {
    return PT(x/c,   y/c  );
  }
};
double dot(PT p, PT q)
```

```
{
   return p.x*q.x+p.y*q.y;
}
double dist2(PT p, PT q)
{
   return dot(p-q,p-q);
}
double distPoint(PT p, PT q)
{
   return sqrt(dot(p-q,p-q));
}
double cross(PT p, PT q)
{
   return p.x*q.y-p.y*q.x;
}
//ostream &operator<<(ostream &os, const PT &p)
//{
//    os << "(" << p.x << "," << p.y << ")";
//}
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)
{
   return PT(-p.y,p.x);
}
PT RotateCW90(PT p)
{
   return PT(p.y,-p.x);
}
PT RotateCCW(PT p, double t)
{
   return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}
PT RotateCW(PT p, double t)
{
   return PT(p.x*cos(t)+p.y*sin(t), -p.x*sin(t)+p.y*cos(t));
}

// find a point from 'a' through 'b' with
// distance d
// use for better precision
PT PointAlongLine(PT a,PT b,double d)
{
   return a + (((b-a) / sqrt(dot(b-a,b-a))) * d);
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c)
{
   return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c)
{
   double r = dot(b-a,b-a);
   if (fabs(r) < EPS) return a;
   r = dot(c-a, b-a)/r;
   if (r < 0) return a;
   if (r > 1) return b;
   return a + (b-a)*r;
}
```

```
// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c)
{
   return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

///return minimum distance from point p to line AB
double distToLine(PT p, PT A, PT B, PT &c)
{
   double scale = (double)
              (dot(p-A,B-A)) /
              (dot(B-A,B-A));
   c.x = A.x + scale * (B.x - A.x);
   c.y = A.y + scale * (B.y - A.y);
   return distPoint(p, c);
}

///return minimum distance from point p to line segment AB
double distToLineSegment(PT p, PT A, PT B, PT &c)
{
   if (dot(B-A,p-A) < EPS)
   {
      c.x = A.x;
      c.y = A.y;
      return distPoint(p, A);
   }
   if (dot(A-B,p-B) < EPS)
   {
      c.x = B.x;
      c.y = B.y;
      return distPoint(p, B);
   }
   return distToLine(p, A, B, c);
}

bool isPointOnSegment(PT p,PT a,PT b)
{
   if(fabs(cross(p-b,a-b))<EPS)
   {
      if(p.x<min(a.x,b.x)||p.x>max(a.x,b.x)) return false;
      if(p.y<min(a.y,b.y)||p.y>max(a.y,b.y)) return false;
      return true;
   }
   return false;
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d)
{
   return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d)
{
   return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d)
{
```

```
   return LinesParallel(a, b, c, d)
        && fabs(cross(a-b, a-c)) < EPS
        && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d)
{
   if (LinesCollinear(a, b, c, d))
   {
      if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
            dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
      if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
         return false;
      return true;
   }
   if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
   if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
   return true;
}
// check if two lines are same
bool areLinesSame(PT a, PT b, PT c, PT d)
{
   if(fabs(cross(a-c,c-d))<EPS && fabs(cross(b-c,c-d))<EPS) return true;
   return false;
}
// check if two lines are parallel
bool areLinesParallel(PT a, PT b, PT c, PT d)
{
   if(fabs(cross(a-b,c-d))<EPS) return true;
   return false;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
/**
this sometimes does not work
//PT ComputeLineIntersection(PT a, PT b, PT c, PT d)
//{
//   b=b-a;
//   d=c-d;
//   c=c-a;
//   return a + b*cross(c, d)/cross(b, d);
//}
*/
PT ComputeLineIntersection(PT a, PT b, PT c, PT d)
{
   double a1,b1,c1,a2,b2,c2;
   a1 = a.y - b.y;
   b1 = b.x - a.x;
   c1 = cross(a, b);
   a2 = c.y - d.y;
   b2 = d.x - c.x;
   c2 = cross(c, d);
   double D = a1 * b2 - a2 * b1;
   return PT((b1 * c2 - b2 * c1) / D,(c1 * a2 - c2 * a1) / D);
}

// compute center of circle given three points
```

```
PT ComputeCircleCenter(PT a, PT b, PT c)
{
   b=(a+b)/2;
   c=(a+c)/2;
   return ComputeLineIntersection(b, b+RotateCW90(a-b), c,
c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q)
{
   bool c = 0;
   int s=p.size();
   for (int i = 0,j=s-1; i < s; j=i++)
   {
      if ( ( (p[i].y > q.y) != (p[j].y > q.y ) ) &&
            (q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y)))
         c = !c;
   }
   return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q)
{
   int s=p.size();
   for (int i = 0,j=s-1; i < s; j=i++)
      if (isPointOnSegment(q,p[j],p[i]))
         return true;
   return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r)
{
   vector<PT> ret;
   b = b-a;
   a = a-c;
   double A = dot(b, b);
   double B = dot(a, b);
   double C = dot(a, a) - r*r;
   double D = B*B - A*C;
   if (D < -EPS) return ret;
   ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
   if (D > EPS)
      ret.push_back(c+a+b*(-B-sqrt(D))/A);
   return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R)
{
   vector<PT> ret;
   double d = sqrt(dist2(a, b));
```

```
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}

// This code computes the area or centroid of a (possibly non-convex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p)
{
  double area = 0;
  for(int i = 0; i < p.size(); i++)
  {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

double ComputeArea(const vector<PT> &p)
{
  return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p)
{
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++)
  {
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p)
{
  for (int i = 0; i < p.size(); i++)
  {
    for (int k = i+1; k < p.size(); k++)
    {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}
/**
Return a parallel line of line ab in counterclockwise
direction with d distance from ab
*/
```

```
pair<PT,PT> getParallelLine(PT a,PT b,double d)
{
   return mp(PointAlongLine(a,RotateCCW90(b-
a)+a,d),PointAlongLine(b,RotateCW90(a-b)+b,d));
}

/**
Return a tangent line of line ab which intersects
with it at point c in counterclockwise direction
*/
pair<PT,PT> getTangentLine(PT a,PT b,PT c)
{
   return mp(RotateCCW90(a-c)+c,RotateCCW90(b-c)+c);
}

vector<PT> halfPlaneIntersection(const vector<PT> &poly, pair<PT,PT> ln)
{
   vector<PT> ret;
   int s=SZ(poly);
   for(int i=0;i<s;i++)
   {
      double c1=cross(ln.sc-ln.fs,poly[i]-ln.fs);
      double c2=cross(ln.sc-ln.fs,poly[(i+1)%s]-ln.fs);
      if(EQ(c1)>=0) ret.psb(poly[i]);
      if(EQ(c1*c2)<0)
      {
         if(!areLinesParallel(poly[i],poly[(i+1)%s],ln.fs,ln.sc))
         {

ret.psb(ComputeLineIntersection(poly[i],poly[(i+1)%s],ln.fs,ln.sc));
         }
      }
   }
   return ret;
}

/*
void Test()
{

   // expected: (-5,2)
   cerr << RotateCCW90(PT(2,5)) << endl;

   // expected: (5,-2)
   cerr << RotateCW90(PT(2,5)) << endl;

   // expected: (-5,2)
   cerr << RotateCCW(PT(2,5),PI/2) << endl;

   // expected: (5,2)
   cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;

   // expected: (5,2) (7.5,3) (2.5,1)
   cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << " "
      << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << " "
      << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;

   // expected: 6.78903
   cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;

   // expected: 1 0 1
   cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
```

```
    << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
    << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

  // expected: 0 0 1
  cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
    << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
    << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

  // expected: 1 1 1 0
  cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;

  // expected: (1,2)
  cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) <<
endl;

  // expected: (1,1)
  cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;


  vector<PT> v;
  v.push_back(PT(0,0));
  v.push_back(PT(5,0));
  v.push_back(PT(5,5));
  v.push_back(PT(0,5));

  // expected: 1 1 1 0 0
  cerr << PointInPolygon(v, PT(2,2)) << " "
    << PointInPolygon(v, PT(2,0)) << " "
    << PointInPolygon(v, PT(0,2)) << " "
    << PointInPolygon(v, PT(5,2)) << " "
    << PointInPolygon(v, PT(2,5)) << endl;

// expected: 0 1 1 1 1
cerr << PointOnPolygon(v, PT(2,2)) << " "
    << PointOnPolygon(v, PT(2,0)) << " "
    << PointOnPolygon(v, PT(0,2)) << " "
    << PointOnPolygon(v, PT(5,2)) << " "
    << PointOnPolygon(v, PT(2,5)) << endl;

// expected: (1,6)
//         (5,4) (4,5)
//         blank line
//         (4,5) (5,4)
//         blank line
//         (4,5) (5,4)
vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
cerr << endl;
u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
```

```
    cerr << endl;
    u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
    for (int i = 0; i < u.size(); i++) cerr << u[i],cerr << " ";
    cerr << endl;

    // area should be 5.0
    // centroid should be (1.1666666, 1.166666)
    PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
    vector<PT> p(pa, pa+4);
    PT c = ComputeCentroid(p);
    cerr << "Area: " << ComputeArea(p) << endl;
    cerr << "Centroid: " << c << endl;


    PT a=PT(1.3,2.6), b=PT(8.1,13.7);
    double d=3.17096;
    PT r=PointAlongLine(a,b,d);
    deb(r.x,r.y);

    return ;
}
*/
/**
////////////////////////////////////////////
line segment intersection for ll value
*/
/**
typedef pair<double,double> pdd;
struct PT
{
    ll x,y;
    PT(){}
    PT(ll x,ll y):x(x),y(y){}
    PT operator + (const PT &p)  const
    {
        return PT(x+p.x, y+p.y);
    }
    PT operator - (const PT &p)  const
    {
        return PT(x-p.x, y-p.y);
    }
    PT operator * (double c)     const
    {
        return PT(x*c,   y*c  );
    }
    PT operator / (double c)     const
    {
        return PT(x/c,   y/c  );
    }
};

PT MV(PT a,PT b){ PT r; r.x = b.x-a.x; r.y = b.y-a.y; return r;}
ll CV(PT a,PT b){return a.x*b.y - a.y*b.x;}

ll dot(PT p, PT q)
{
    return p.x*q.x+p.y*q.y;
}
ll cross(PT p, PT q)
{
    return p.x*q.y-p.y*q.x;
}
```

```
///***
bool areLinesSame(PT a, PT b, PT c, PT d)
{
   if(cross(a-c,c-d)==0 && cross(b-c,c-d)==0) return true;
   return false;
}
///***
bool areLinesParallel(PT a, PT b, PT c, PT d)
{
   if(cross(a-b,c-d)==0) return true;
   return false;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
//void ComputeLineIntersection(PT a, PT b, PT c, PT d,pdd &ret)
//{
//    b=b-a;
//    d=c-d;
//    c=c-a;
//    double h=(double)cross(c, d)/(double)cross(b, d);
//    ret.xx=(double) a.x + (double) b.x * h;
//    ret.yy=(double) a.y + (double) b.y * h;
//    return ;
//}

PT ComputeLineIntersection(PT a, PT b, PT c, PT d,pdd &ret)
{
   double a1,b1,c1,a2,b2,c2;
   a1 = a.y - b.y;
   b1 = b.x - a.x;
   c1 = cross(a, b);
   a2 = c.y - d.y;
   b2 = d.x - c.x;
   c2 = cross(c, d);
   double D = a1 * b2 - a2 * b1;
   ret=mp((b1 * c2 - b2 * c1) / D,(c1 * a2 - c2 * a1) / D);
   return ret;
}

bool onsegment(PT a,PT b,PT c){
   return ( min(a.x,b.x)<=c.x && c.x<=max(a.x,b.x) && min(a.y,b.y)<=c.y
&& c.y<=max(a.y,b.y) ) ;
}
bool isSegmentIntersect(PT p1,PT p2,PT p3,PT p4)
{
   ll d1,d2,d3,d4;

   d1 = CV(MV(p3,p4),MV(p3,p1));
   d2 = CV(MV(p3,p4),MV(p3,p2));
   d3 = CV(MV(p1,p2),MV(p1,p3));
   d4 = CV(MV(p1,p2),MV(p1,p4));

   int s1,s2,s3,s4;
   s1=d1==0?0:d1<0?-1:1;
   s2=d2==0?0:d2<0?-1:1;
   s3=d3==0?0:d3<0?-1:1;
   s4=d4==0?0:d4<0?-1:1;

   if(s1*s2<0 && s3*s4<0)  return true;
```

```
   if(!d1 && onsegment(p3,p4,p1))  return true;
   if(!d2 && onsegment(p3,p4,p2))  return true;
   if(!d3 && onsegment(p1,p2,p3))  return true;
   if(!d4 && onsegment(p1,p2,p4))  return true;

   return false;
}
*/


int main()
{
   //Test();
   return 0;
}
```

125. Geometry 3D (Rumman Bhai)

```
#define zero(x) (((x)>0?(x):-(x))<EPS)

const double INF = 1e100;
const double EPS = 1e-9;

int EQ(double x)
{
   if(fabs(x)<EPS) return 0;
   else if(x>0) return 1;
   else return -1;
}

struct point3
{
   double x,y,z;
   point3(){}
   point3(double x,double y,double z):x(x),y(y),z(z){}
   point3 operator + (const point3 &p)  const
   {
      return point3(x+p.x, y+p.y, z+p.z);
   }
   point3 operator - (const point3 &p)  const
   {
      return point3(x-p.x, y-p.y, z-p.z);
   }
   point3 operator * (double c)    const
   {
      return point3(x*c, y*c , z*c);
   }
   point3 operator / (double c)    const
   {
      return point3(x/c, y/c, z/c);
   }
};
struct line3
{
   point3 a,b;
   line3(){}
   line3(point3 a,point3 b):a(a),b(b){}
};
struct plane3
{
   point3 a,b,c;
   plane3(){}
```

```
    plane3(point3 a,point3 b,point3 c):a(a),b(b),c(c){}
};

//compute cross product U x V
point3 xmult(point3 u,point3 v){
    point3 ret;
        ret.x=u.y*v.z-v.y*u.z;
        ret.y=u.z*v.x-u.x*v.z;
        ret.z=u.x*v.y-u.y*v.x;
        return ret;
}

//compute dot product U . V
double dmult(point3 u,point3 v){
        return u.x*v.x+u.y*v.y+u.z*v.z;
}

// Vector difference U - V
point3 subt(point3 u,point3 v){
        point3 ret;
        ret.x=u.x-v.x;
        ret.y=u.y-v.y;
        ret.z=u.z-v.z;
        return ret;
}

// Vector addition U + V
point3 addt(point3 u,point3 v){
        point3 ret;
        ret.x=u.x+v.x;
        ret.y=u.y+v.y;
        ret.z=u.z+v.z;
        return ret;
}

// Take the plane normal vector
point3 pvec(plane3 s){
        return xmult(subt(s.a,s.b),subt(s.b,s.c));
}
point3 pvec(point3 s1,point3 s2,point3 s3){
        return xmult(subt(s1,s2),subt(s2,s3));
}

// Distance between two points, the size of a single parameter of the
alignment amount
double distance(point3 p1,point3 p2){
        return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-
p2.z)*(p1.z-p2.z));
}

// Vector magnitude
double vlen(point3 p){
        return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
}

// Sentenced collinear
int dots_inline(point3 p1,point3 p2,point3 p3){
        return vlen(xmult(subt(p1,p2),subt(p2,p3)))<EPS;
}

// Sentenced to four points are coplanar
int dots_onplane(point3 a,point3 b,point3 c,point3 d){
```

```
        return zero(dmult(pvec(a,b,c),subt(d,a)));
}

// Sentenced point if the line segment, inclusive and collinear
int dot_online_in(point3 p,line3 l){
        return zero(vlen(xmult(subt(p,l.a),subt(p,l.b))))&&(l.a.x-
p.x)*(l.b.x-p.x)<EPS&&
                (l.a.y-p.y)*(l.b.y-p.y)<EPS&&(l.a.z-p.z)*(l.b.z-p.z)<EPS;
}
int dot_online_in(point3 p,point3 l1,point3 l2){
        return zero(vlen(xmult(subt(p,l1),subt(p,l2))))&&(l1.x-p.x)*(l2.x-
p.x)<EPS&&
                (l1.y-p.y)*(l2.y-p.y)<EPS&&(l1.z-p.z)*(l2.z-p.z)<EPS;
}

// Sentenced point on whether the line segment, not inclusive
int dot_online_ex(point3 p,line3 l){
        return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-
l.a.y)||!zero(p.z-l.a.z))&&
                (!zero(p.x-l.b.x)||!zero(p.y-l.b.y)||!zero(p.z-l.b.z));
}
int dot_online_ex(point3 p,point3 l1,point3 l2){
        return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-
l1.y)||!zero(p.z-l1.z))&&
                (!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));
}

// Determines whether a point on a triangular space, including borders,
collinear meaningless
int dot_inplane_in(point3 p,plane3 s){
        return zero(vlen(xmult(subt(s.a,s.b),subt(s.a,s.c)))-
vlen(xmult(subt(p,s.a),subt(p,s.b)))-
                vlen(xmult(subt(p,s.b),subt(p,s.c)))-
vlen(xmult(subt(p,s.c),subt(p,s.a))));
}
int dot_inplane_in(point3 p,point3 s1,point3 s2,point3 s3){
        return zero(vlen(xmult(subt(s1,s2),subt(s1,s3)))-
vlen(xmult(subt(p,s1),subt(p,s2)))-
                vlen(xmult(subt(p,s2),subt(p,s3)))-
vlen(xmult(subt(p,s3),subt(p,s1))));
}

// Determines whether a point on a triangular space, not including
borders, collinear meaningless
int dot_inplane_ex(point3 p,plane3 s){
        return
dot_inplane_in(p,s)&&vlen(xmult(subt(p,s.a),subt(p,s.b)))>EPS&&

        vlen(xmult(subt(p,s.b),subt(p,s.c)))>EPS&&vlen(xmult(subt(p,s.c),
subt(p,s.a)))>EPS;
}
int dot_inplane_ex(point3 p,point3 s1,point3 s2,point3 s3){
        return
dot_inplane_in(p,s1,s2,s3)&&vlen(xmult(subt(p,s1),subt(p,s2)))>EPS&&

        vlen(xmult(subt(p,s2),subt(p,s3)))>EPS&&vlen(xmult(subt(p,s3),su
bt(p,s1)))>EPS;
}

// Sentenced to two line segments on the same side, returns 0 point line
segment, are not coplanar meaningless
```

```
int same_side(point3 p1,point3 p2,line3 l){
	return
dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))>EP
S;
}
int same_side(point3 p1,point3 p2,point3 l1,point3 l2){
	return
dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))>EPS;
}

// Sentenced to two different sides of the line segment, returns 0 point
line segment, are not coplanar meaningless
int opposite_side(point3 p1,point3 p2,line3 l){
	return
dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))<-
EPS;
}
int opposite_side(point3 p1,point3 p2,point3 l1,point3 l2){
	return
dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))<-EPS;
}

// Sentenced to two points in the plane on the same side, point in the
plane returns 0
int same_side(point3 p1,point3 p2,plane3 s){
	return
dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))>EPS;
}
int same_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
	return
dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))>EPS;
}

// Sentenced to two points in the plane of the opposite side, the point in
the plane returns 0
int opposite_side(point3 p1,point3 p2,plane3 s){
	return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))<-
EPS;
}
int opposite_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
	return
dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))<-
EPS;
}

// Sentenced to two parallel lines
int parallel(line3 u,line3 v){
	return vlen(xmult(subt(u.a,u.b),subt(v.a,v.b)))<EPS;
}
int parallel(point3 u1,point3 u2,point3 v1,point3 v2){
	return vlen(xmult(subt(u1,u2),subt(v1,v2)))<EPS;
}

// Sentenced to two plane-parallel
int parallel(plane3 u,plane3 v){
	return vlen(xmult(pvec(u),pvec(v)))<EPS;
}
int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
	return vlen(xmult(pvec(u1,u2,u3),pvec(v1,v2,v3)))<EPS;
}

// Sentence straight and parallel to the plane
```

```
int parallel(line3 l,plane3 s){
	return zero(dmult(subt(l.a,l.b),pvec(s)));
}
int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
	return zero(dmult(subt(l1,l2),pvec(s1,s2,s3)));
}

// Sentenced to two straight lines perpendicular
int perpendicular(line3 u,line3 v){
	return zero(dmult(subt(u.a,u.b),subt(v.a,v.b)));
}
int perpendicular(point3 u1,point3 u2,point3 v1,point3 v2){
	return zero(dmult(subt(u1,u2),subt(v1,v2)));
}

// Sentenced to two planes perpendicular
int perpendicular(plane3 u,plane3 v){
	return zero(dmult(pvec(u),pvec(v)));
}
int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3
v2,point3 v3){
	return zero(dmult(pvec(u1,u2,u3),pvec(v1,v2,v3)));
}

// Sentence straight and parallel to the plane
int perpendicular(line3 l,plane3 s){
	return vlen(xmult(subt(l.a,l.b),pvec(s)))<EPS;
}
int perpendicular(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
	return vlen(xmult(subt(l1,l2),pvec(s1,s2,s3)))<EPS;
}
```

```
// Sentenced to two segments intersect, inclusive and partially overlap
int intersect_in(line3 u,line3 v){
	if (!dots_onplane(u.a,u.b,v.a,v.b))
		return 0;
	if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
		return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
	return
dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_o
nline_in(v.b,u);
}
int intersect_in(point3 u1,point3 u2,point3 v1,point3 v2){
	if (!dots_onplane(u1,u2,v1,v2))
		return 0;
	if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
		return
!same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
	return
dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,
u2)||dot_online_in(v2,u1,u2);
}

// Sentenced to two line segments intersect, not inclusive and partially
overlap
int intersect_ex(line3 u,line3 v){
	return
dots_onplane(u.a,u.b,v.a,v.b)&&opposite_side(u.a,u.b,v)&&opposite_sid
e(v.a,v.b,u);
}
int intersect_ex(point3 u1,point3 u2,point3 v1,point3 v2){
```

```
        return
dots_onplane(u1,u2,v1,v2)&&opposite_side(u1,u2,v1,v2)&&opposite_sid
e(v1,v2,u1,u2);
}

// Sentenced triangle intersection and space segments, including cross
the boundary and (in part) that contains
int intersect_in(line3 l,plane3 s){
        return !same_side(l.a,l.b,s)&&!same_side(s.a,s.b,l.a,l.b,s.c)&&

        !same_side(s.b,s.c,l.a,l.b,s.a)&&!same_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_in(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
        return !same_side(l1,l2,s1,s2,s3)&&!same_side(s1,s2,l1,l2,s3)&&
                !same_side(s2,s3,l1,l2,s1)&&!same_side(s3,s1,l1,l2,s2);
}

// Sentenced triangle intersection and space segments, not including
delivery to the boundary and (in part) that contains
int intersect_ex(line3 l,plane3 s){
        return
opposite_side(l.a,l.b,s)&&opposite_side(s.a,s.b,l.a,l.b,s.c)&&

        opposite_side(s.b,s.c,l.a,l.b,s.a)&&opposite_side(s.c,s.a,l.a,l.b,s.b)
;
}
int intersect_ex(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
        return
opposite_side(l1,l2,s1,s2,s3)&&opposite_side(s1,s2,l1,l2,s3)&&

        opposite_side(s2,s3,l1,l2,s1)&&opposite_side(s3,s1,l1,l2,s2);
}

// Calculate the intersection of two straight, pay attention to prejudge
whether coplanar and parallel to the straight line!
// Line intersects the intersection please also sentenced segment (and
still have to determine whether the parallel!)
point3 intersection(line3 u,line3 v){
        point3 ret=u.a;
        double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
                        /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-
v.b.x));
        ret.x+=(u.b.x-u.a.x)*t;
        ret.y+=(u.b.y-u.a.y)*t;
        ret.z+=(u.b.z-u.a.z)*t;
        return ret;
}
point3 intersection(point3 u1,point3 u2,point3 v1,point3 v2){
        point3 ret=u1;
        double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
                        /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
        ret.x+=(u2.x-u1.x)*t;
        ret.y+=(u2.y-u1.y)*t;
        ret.z+=(u2.z-u1.z)*t;
        return ret;
}

// Calculate the intersection of the straight line and the plane, pay
attention to prejudge whether or not parallel, and to ensure that three
non-collinear!
// Line and space triangle intersection please also judge
point3 intersection(line3 l,plane3 s){
```

```
	point3 ret=pvec(s);
	double t=(ret.x*(s.a.x-l.a.x)+ret.y*(s.a.y-l.a.y)+ret.z*(s.a.z-l.a.z))/
		(ret.x*(l.b.x-l.a.x)+ret.y*(l.b.y-l.a.y)+ret.z*(l.b.z-l.a.z));
	ret.x=l.a.x+(l.b.x-l.a.x)*t;
	ret.y=l.a.y+(l.b.y-l.a.y)*t;
	ret.z=l.a.z+(l.b.z-l.a.z)*t;
	return ret;
}
point3 intersection(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
	point3 ret=pvec(s1,s2,s3);
	double t=(ret.x*(s1.x-l1.x)+ret.y*(s1.y-l1.y)+ret.z*(s1.z-l1.z))/
		(ret.x*(l2.x-l1.x)+ret.y*(l2.y-l1.y)+ret.z*(l2.z-l1.z));
	ret.x=l1.x+(l2.x-l1.x)*t;
	ret.y=l1.y+(l2.y-l1.y)*t;
	ret.z=l1.z+(l2.z-l1.z)*t;
	return ret;
}

// Calculate the two planes intersecting line, pay attention to prejudge
whether or not parallel, and to ensure that three non-collinear!
line3 intersection(plane3 u,plane3 v){
	line3 ret;
	ret.a=parallel(v.a,v.b,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):i
ntersection(v.a,v.b,u.a,u.b,u.c);
	ret.b=parallel(v.c,v.a,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):i
ntersection(v.c,v.a,u.a,u.b,u.c);
	return ret;
}
line3 intersection(point3 u1,point3 u2,point3 u3,point3 v1,point3
v2,point3 v3){
	line3 ret;
	ret.a=parallel(v1,v2,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):inters
ection(v1,v2,u1,u2,u3);
	ret.b=parallel(v3,v1,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):inters
ection(v3,v1,u1,u2,u3);
	return ret;
}

// Point to the straight line distance
double ptoline(point3 p,line3 l){
	return vlen(xmult(subt(p,l.a),subt(l.b,l.a)))/distance(l.a,l.b);
}
double ptoline(point3 p,point3 l1,point3 l2){
	return vlen(xmult(subt(p,l1),subt(l2,l1)))/distance(l1,l2);
}

// Point to plane distance
double ptoplane(point3 p,plane3 s){
	return fabs(dmult(pvec(s),subt(p,s.a)))/vlen(pvec(s));
}
double ptoplane(point3 p,point3 s1,point3 s2,point3 s3){
	return fabs(dmult(pvec(s1,s2,s3),subt(p,s1)))/vlen(pvec(s1,s2,s3));
}

// Straight line to straight line distance
double linetoline(line3 u,line3 v){
	point3 n=xmult(subt(u.a,u.b),subt(v.a,v.b));
	return fabs(dmult(subt(u.a,v.a),n))/vlen(n);
}
double linetoline(point3 u1,point3 u2,point3 v1,point3 v2){
	point3 n=xmult(subt(u1,u2),subt(v1,v2));
	return fabs(dmult(subt(u1,v1),n))/vlen(n);
```

```
}

// The angle between two straight lines cos value
double angle_cos(line3 u,line3 v){
	return
dmult(subt(u.a,u.b),subt(v.a,v.b))/vlen(subt(u.a,u.b))/vlen(subt(v.a,v.b));
}
double angle_cos(point3 u1,point3 u2,point3 v1,point3 v2){
	return
dmult(subt(u1,u2),subt(v1,v2))/vlen(subt(u1,u2))/vlen(subt(v1,v2));
}

// The angle between two planes cos value
double angle_cos(plane3 u,plane3 v){
	return dmult(pvec(u),pvec(v))/vlen(pvec(u))/vlen(pvec(v));
}
double angle_cos(point3 u1,point3 u2,point3 u3,point3 v1,point3
v2,point3 v3){
	return
dmult(pvec(u1,u2,u3),pvec(v1,v2,v3))/vlen(pvec(u1,u2,u3))/vlen(pvec(v1,
v2,v3));
}

// Straight plane angle value sin
double angle_sin(line3 l,plane3 s){
	return dmult(subt(l.a,l.b),pvec(s))/vlen(subt(l.a,l.b))/vlen(pvec(s));
}
double angle_sin(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
	return
dmult(subt(l1,l2),pvec(s1,s2,s3))/vlen(subt(l1,l2))/vlen(pvec(s1,s2,s3));
}

int main()
{
  return 0;
}
```

126. Geometry 2D (Ovishek)

```
#define EPS 1e-9
#define pi acos(-1.0)
int cmp(double x)
{
  if(fabs(x) < EPS) return 0;
  return x < 0 ? -1 : 1;
}
struct PT{
  double x, y;
  PT(){x = y = 0; }
  PT(double _x, double _y) {x = _x, y = _y; }
  PT operator-(const PT &a) const{
    return PT(x - a.x, y - a.y);
  }
  PT operator+(const PT &a) const{
    return PT(x + a.x, y + a.y);
  }
  PT operator*(double a) const{
    return PT(x * a, y * a);
  }
  PT operator/(double a) const{
    return PT(x / a, y / a);
  }
```

```
    double val()
    {
        return sqrt(x * x + y * y);
    }
    PT unit()
    {
        return (*this) / val();
    }
    void scan()
    {
        scanf("%lf %lf", &x, &y);
    }
    void print()
    {
        printf("(%.4f, %.4f)", x, y);
    }
};
struct line{
    double a, b, c;
};

double dist(PT a, PT b)
{
    return (a - b).val();
}

double dist2(PT a, PT b)
{
    a = a - b;
    return a.x * a.x + a.y * a.y;
}

double dot(PT a, PT b)
{
    return a.x * b.x + a.y * b.y;
}
double cross(PT a, PT b)
{
    return a.x * b.y - a.y * b.x;
}
PT RotateCCW90(PT p)
{
    return PT(-p.y,p.x);
}
PT RotateCW90(PT p)
{
    return PT(p.y,-p.x);
}
PT RotateCCW(PT p, double t)
{
    return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}
PT RotateCW(PT p, double t)
{
    return PT(p.x*cos(t)+p.y*sin(t), -p.x*sin(t)+p.y*cos(t));
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c)
{
    double r = dot(b-a,b-a);
    if (fabs(r) < EPS) return a;
    r = dot(c-a, b-a)/r;
```

```
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c)
{
    return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}
// returns bisector of angle YXZ
line bisector(PT Y, PT X, PT Z)
{
    PT xy = (Y - X)/(Y - X).val();
    PT xz = (Z - X)/(Z - X).val();
    PT d = xy + xz;
    line ret{d.y, -d.x, d.x * X.y - d.y * X.x};
    return ret;
}

vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r)
{
    vector<PT>ret;
    b=b-a;
    a=a-c;

    double A=dot(b, b);
    double B=dot(a, b);
    double C=dot(a, a)-r*r;
    double D=B*B-A*C;
    if(D<-EPS) return ret;

    ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);

    if(D>EPS) ret.push_back(c+a+b*(-B-sqrt(D))/A);

    return ret;
}
PT ComputeLineIntersection(PT a, PT b, PT c, PT d)
{
    double a1=a.y-b.y;
    double b1=b.x-a.x;
    double c1=cross(a, b);
    double a2=c.y-d.y;
    double b2=d.x-c.x;
    double c2=cross(c, d);
    double D=a1*b2-a2*b1;

    return PT((b1*c2-b2*c1)/D, (c1*a2-c2*a1)/D);
}
const double big = 100000000000000000.0;
vector<PT> CircleTouchingPoints(PT c, double r, PT a)
{
    double d = dist(c, a);
    double angle = asin(r/d);
    double length = sqrt(d*d - r*r);
    PT ac = c-a;
    ac = RotateCCW(ac, angle);
    ac = ac / ac.val() * length;
    vector<PT> ret;
    ret.push_back(a+ac);
```

```
    ac = c-a;
    ac = RotateCCW(ac, -angle);
    ac = ac / ac.val() * length;
    ret.push_back(ac + a);
    return ret;
}

double CircleArcDistance(PT c, double r, PT a, PT b)
{
    double d = dist(a, b);
    double angle = acos((2.0*r*r - d*d) / (2.0*r*r));
    return r * angle;
}
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R)
{
    vector<PT> ret;
    double d = sqrt(dist2(a, b));
    if (d > r+R || d+min(r, R) < max(r, R)) return ret;
    double x = (d*d-R*R+r*r)/(2*d);
    double y = sqrt(r*r-x*x);
    PT v = (b-a)/d;
    ret.push_back(a+v*x + RotateCCW90(v)*y);
    if (y > 0)
        ret.push_back(a+v*x - RotateCCW90(v)*y);
    return ret;
}
int main()
{
    int t;
    scanf("%d", &t);
    while(t--)
    {
        PT a, b, c;
        double r;
        a.scan();
        b.scan();
        scanf("%lf", &r);
        c = PT(0, 0);
        double ans = big;
        if(DistancePointSegment(a,b,c) >= r) ans = dist(a, b);
        auto vtA = CircleTouchingPoints(c, r, a);
        auto vtB = CircleTouchingPoints(c, r, b);
        for(int i = 0; i<2; i++)
        {
            for(int j = 0; j<2; j++)
            {
                double d = dist(a, vtA[i]) + dist(b, vtB[j]) + CircleArcDistance(c, r,
vtA[i], vtB[j]);
                ans = min(ans, d);
            }
        }
        printf("%.3f\n", ans);
    }
    return 0;
}
```

127. Geometry 3D (Ovishek)

```
#define EPS 1e-9
#define pi acos(-1.0)
int cmp(double x)
{
    if(fabs(x) < EPS) return 0;
```

```
    return x < 0 ? -1 : 1;
}
struct PT{
    double x, y, z;
    PT(){x = y = 0; }
    PT(double _x, double _y, double _z) {x = _x, y = _y, z = _z; }
    PT operator-(const PT &a) const{
        return PT(x - a.x, y - a.y, z - a.z);
    }
    PT operator+(const PT &a) const{
        return PT(x + a.x, y + a.y, z + a.z);
    }
    PT operator*(double a) const{
        return PT(x * a, y * a, z * a);
    }
    PT operator/(double a) const{
        return PT(x / a, y / a, z / a);
    }
    PT unit()
    {
        return (*this) / val();
    }
    double val()
    {
        return sqrt(x * x + y * y + z * z);
    }
    void scan()
    {
        scanf("%lf %lf %lf", &x, &y, &z);
    }
    void print()
    {
        printf("(%.4f, %.4f, %.4f)", x, y, z);
    }
};

double dist(PT a, PT b)
{
    return (a - b).val();
}

double dist2(PT a, PT b)
{
    a = a - b;
    return a.x * a.x + a.y * a.y + a.z * a.z;
}
double dot(PT a, PT b)
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
PT cross(PT a, PT b)
{
    return PT(a.y * b.z - b.y * a.z, b.x * a.z - a.x * b.z, a.x * b.y - a.y * b.x);
}
double DistancePointSegment(PT a, PT b, PT c)
{
    //a.print(); b.print(); c.print();
    //cout << endl;
    //return dist(a, c);
    if(cmp(dot(b-a, c-a)) < 0) return dist(a, c);
    if(cmp(dot(a-b, c-b)) < 0) return dist(b, c);
    return fabs(cross((b-a).unit(), c-a).val());
```

```
}
double TriArea(PT a, PT b, PT c)
{
   return 0.5 * fabs(cross(b-a, c-a).val());
}
double TriToPointDistance(PT a, PT b, PT c, PT d)
{
   PT norm = cross(b-a, c-a).unit();
   PT ana = cross(b-a, norm);
   int f = 0;
   if(cmp(dot(ana, d-a)) > 0) f = 1;
   ana = cross(c-b, norm);
   if(cmp(dot(ana, d-b)) > 0) f = 1;
   ana = cross(a-c, norm);
   if(cmp(dot(ana, d-c)) > 0) f = 1;
   if(f==0) {
      PT ad = d - a;
      return fabs(dot(ad, norm));
   } else{
      return min(DistancePointSegment(a, b, d),
min(DistancePointSegment(b, c, d), DistancePointSegment(c, a, d)));
   }
}
const double big = 1000000000000000.0;
double TriToSegmentDistance(PT a, PT b, PT c, PT d, PT e)
{
   double l = 0.0, r = 1.0;
   int cnt = 50;
   double ret = big;
   while(cnt--)
   {
      double mid1 = l + (r-l)/3.0, mid2 = r - (r-l)/3.0;
      double x = TriToPointDistance(a, b, c, d + (e-d)*mid1);
      double y = TriToPointDistance(a, b, c, d + (e-d)*mid2);
      if(x < y){
         r = mid2;
         ret = x;
      } else{
         ret = y;
         l = mid1;
      }
   }
   return ret;
}

double TriToTriDistance(PT a, PT b, PT c, PT d, PT e, PT f)
{
   double ret = big;
   ret = min(ret, TriToSegmentDistance(a, b, c, d, e));
   ret = min(ret, TriToSegmentDistance(a, b, c, e, f));
   ret = min(ret, TriToSegmentDistance(a, b, c, f, d));
   ret = min(ret, TriToSegmentDistance(d, e, f, a, b));
   ret = min(ret, TriToSegmentDistance(d, e, f, b, c));
   ret = min(ret, TriToSegmentDistance(d, e, f, c, a));
   return ret;
}

int main()
{
   int t;
   scanf("%d", &t);
   while(t--)
```

```
    {
        int n = 4;
        vector<PT> vtA(4), vtB(4);
        for(int i = 0; i<n; i++)
            vtA[i].scan();
        for(int i = 0; i<n; i++)
            vtB[i].scan();
        double ans = big;
        for(int i = 0; i<n; i++)
        for(int j = i+1; j<n; j++)
        for(int k = j+1; k<n; k++){
            PT a = vtA[i], b = vtA[j], c = vtA[k];
            for(int _i = 0; _i<n; _i++)
            for(int _j = _i+1; _j<n; _j++)
            for(int _k = _j+1; _k<n; _k++){
                PT d = vtB[_i], e = vtB[_j], f = vtB[_k];
                ans = min(ans, TriToTriDistance(a, b, c, d, e, f));
            }
        }
        printf("%.2f\n", ans);
    }
    return 0;
}
```

128. Circle Union

```
const double EPS = 1e-8;
const double PI  = acos(-1.0);
const double TAU = 2.0 * PI;
const double INF = 1e99;
int sig(double x) {
    return x < -EPS ? -1 : x > EPS ? 1 : 0;
}
template<class T> T pow2(T x) {
    return x * x;
}
class Vector {
public:
    double x, y;
    Vector() {}
    Vector(double x, double y): x(x), y(y) {}

    Vector operator -() const {
        return Vector(-x, -y);
    }
    Vector operator +(const Vector &v) const {
        return Vector(x+v.x, y+v.y);
    }
    Vector operator -(const Vector &v) const {
        return Vector(x-v.x, y-v.y);
    }
    Vector operator *(const double &s) const {
        return Vector(x * s, y * s);
    }
    Vector operator /(const double &s) const {
        return Vector(x / s, y / s);
    }
    double operator *(const Vector &v) const {
        return x*v.x + y*v.y;
    }
    double operator ^(const Vector &v) const {
        return x*v.y - y*v.x;
    }
```

```
    // rotate vector (Right/Left hand)
    Vector  R(double co, double si) {
        return Vector(x*co-y*si, y*co+x*si);
    }
    Vector  L(double co, double si) {
        return Vector(x*co+y*si, y*co-x*si);
    }
    Vector  R(double th) {
        return R(cos(th), sin(th));
    }
    Vector  L(double th) {
        return L(cos(th), sin(th));
    }

    double len2() {
        return x*x + y*y;
    }
    double len()  {
        return sqrt(len2());
    }
    double ang()  {
        return atan2(y, x);   // angle of vector
    }
    Vector e(double s = 1.0) {
        return *this / len() * s;
    }
};
typedef Vector Point;
```

```
class Line {
public:
    Point a, b;
    Line() {}
    Line(Point a, Point b): a(a), b(b) {}
};

class Circle {
public:
    Point o;
    double r;
    Circle() {}
    Circle(Point o, double r): o(o), r(r) {}

    int posi(Circle c) {
        double d = (o - c.o).len();
        int in = sig(d - fabs(r - c.r)), ex = sig(d - (r + c.r));
        return in<0 ? -2 : in==0? -1 : ex==0 ? 1 : ex>0? 2 : 0;
    }
    Line chord(Circle c) {
        Vector v = c.o - o;
        double co = (pow2(r) + v.len2() - pow2(c.r)) / (2 * r * v.len());
        double si = sqrt(fabs(1.0 - pow2(co)));
        return Line(v.L(co, si).e(r) + o, v.R(co, si).e(r) + o);
    }
};
struct Range {
    double t;
    int evt;
    Point p;
    Range() {}
```

```
    Range(double t, int evt, Point p) : t(t), evt(evt), p(p) {}

    bool operator <(const Range &s) const {
        return sig(t - s.t) < 0 || (sig(t - s.t) == 0 && evt > s.evt);
    }
};
const int MAX_N = 1000 + 10;
Circle C[MAX_N];
Range R[MAX_N<<1];
// sort circle with desending of radii
bool cmp_r(const Circle &a, const Circle &b) {
    return a.r > b.r;
}

double segment_area(double r, double t) {
    return pow2(r) * (t - sin(t)) / 2;
}
double union_circle(Circle C[], int &n) {
    sort(C, C + n, cmp_r);
    int k = 0;
    for (int i = 0; i < n; i++) {
        if (sig(C[i].r) == 0) break;
        int j = 0;
        for (j = 0; j < k; j++)
            if (C[i].posi(C[j]) < 0 || !sig((C[i].o - C[j].o).len()))
                break;
        if (j == k)
            C[k++] = C[i];
    }
    n = k;
    double ans = 0;
    for (int i = 0; i < n; ++ i) {
        Point mpi = Point(- C[i].r, 0.0) + C[i].o;
        int nc = 0, rcnt = 0;
        R[rcnt++] = Range(-PI,  1, mpi);
        R[rcnt++] = Range( PI, -1, mpi);
        for (int j = 0; j < n; ++ j) {
            if (j == i || C[i].posi(C[j])) continue;

            Line l = C[i].chord(C[j]);
            double jR = (l.a - C[i].o).ang(), jL = (l.b - C[i].o).ang();

            if (sig(jR - jL) > 0) ++ nc;
            R[rcnt++] = Range(jR,  1, l.a);
            R[rcnt++] = Range(jL, -1, l.b);
        }
        sort(R, R + rcnt);
        double pj = - PI;
        Point  pp = mpi;
        for(int j = 0; j < rcnt; ++ j) {
            nc += R[j].evt;
            if((nc == 2 && R[j].evt > 0) || nc == 0)
                    ans += segment_area(C[i].r, R[j].t - pj) + (pp ^ R[j].p) / 2;
            pj = R[j].t;
            pp = R[j].p;
        }
    }
    return ans;
}

int main() {
```

```
    int n;
    while (scanf("%d", &n) == 1) {
        if(n == 0) break;
        for (int i = 0; i < n; i++) scanf("%lf%lf%lf", &C[i].o.x, &C[i].o.y, &C[i].r);

        double ans = union_circle(C, n);
        printf("%.3f\n", ans);
    }
    return 0;
}
```