Index Page

Graph And Network Flows and BPM

Data Structure

Geometry

Number Theory + Equations

```
//Model Code
//#pragma comment(linker, "/STACK:60000000")
#include <iostream>       #include <stdio.h>
#include <string>         #include <stdlib.h>
#include <map>            #include <string.h>
#include <queue>          #include<math.h>
#include <stack>          #include <vector>
#include <algorithm>      #include <set>
#include <list>           #include <cstring>
#include <sstream>

using namespace std;

typedef long longll;
typedef pair<int,int>pii;

#define ERR 1e-9
#define PI 3.141592653589793
#define FOREACH(it,x) for(__typeof((x).begin()) it=(x.begin());it!=(x).end();++it)

#define MP(a,b) make_pair(a,b)
#define Clear(x,with) memset(x,with,sizeof(x))
#define SZ(x) (int)x.size()
#define pbpush_back
#define popcount(i) __builtin_popcount(i)
#define two(X) (1<<(X))
#define twoL(X) (((ll)(1))<<(X))
#define contain(S,X) (((S)&two(X))!=0)
#define fs first
#define sc second
#define EQ(a,b) (fabs((a)-(b))<ERR)
#define Unique(store)
            store.resize(unique(store.begin(),store.end())-store.begin());
//For debugging
#define debug_array(a,n) for(inti=0;i<n;i++) cerr<<a[i]<<" "; cerr<<endl;
#define debug(args...) {dbg,args; cerr<<endl;}
struct debugger{template<typename T> debugger& operator ,
                  (const T& v){cerr<<v<<"\t"; return *this; }}dbg;

//Important Functions
template<class T> string toString(T n)
      {ostringstreamoss;oss<<n;oss.flush();return oss.str();}

inttoInt(string s){int r=0;istringstream sin(s);sin>>r;return r;}
intBigMod(llB,llP,ll M){  ll R=1;
            while(P>0) {if(P%2==1){R=(R*B)%M;}P/=2;B=(B*B)%M;} return (int)R;}

//int dx[]={1,0,-1,0};intdy[]={0,1,0,-1}; //4 Direction
//int dx[]={1,1,0,-1,-1,-1,0,1};intdy[]={0,1,1,1,0,-1,-1,-1};//8 direction
//int dx[]={2,1,-1,-2,-2,-1,1,2};intdy[]={1,2,2,1,-1,-2,-2,-1};//Knight Direction
//int dx[]={-1,-1,+0,+1,+1,+0};intdy[]={-1,+1,+2,+1,-1,-2}; //Hexagonal Direction

#define INF (1<<28)
#define SIZE 1100
int main()
{
      return 0;
}
```

//Articulaton Point

```
vector<int>adj[SIZE];
int t,root;
bool color[SIZE],flag[SIZE]; //flag to mark
//articulation points.
int par[SIZE],low[SIZE],tme[SIZE];

int dfs(int u)
{
    color[u] = 1;
    tme[u] = low[u] = t++;

    int i,subtree=0,v;
    for(i=0;i<SZ(adj[u]);i++) {
        v = adj[u][i];
        if(!color[v]) {
            subtree++; par[v] = u;
            low[u] = min(low[u],dfs(v));

            if(low[v]>=tme[u]&&u!=root) flag[u] = 1;
        }
        else if(v!=par[u]) {
            low[u] = min(low[u],tme[v]);
        }
    }
    if(u==root && subtree>1) flag[u] = 1;

    return low[u];
}
```

//Disjoint Set(Union Find)

```
struct edges {

    int u,v,cost;
};
edges edge[125000];
int par[400],rank[400],deg[400];

bool comp(edges p,edges q){
    return p.cost<q.cost;
}
int find_par(int u) {
    if(par[u]!=u) par[u] = find_par(par[u]);
    return par[u];
}

void Union(inta,int b) {
    if(rank[a]>rank[b])
    { rank[a]++; par[b] = a; }
    else { rank[b]++; par[a] = b; }
}
```

//SCC(Strongly Conn. Compo.)

```
vector<int>arr,adj[SIZE],trans[SIZE],graph[SIZE];
int color[SIZE],nodes,N,par[SIZE];

void dfs1(int u) {
    color[u] = 1;

    for(int i=0;i<SZ(adj[u]);i++) {
        int v = adj[u][i];
        if(!color[v])   dfs1(v);
    }
    arr.pb(u);
}
void dfs2(int u,int t) {
    color[u] = t;

    for(int i=0;i<SZ(trans[u]);i++) {
        int v = trans[u][i];
        if(!color[v]) dfs2(v,t);
    }
}

void Rebuild_graph() {
    int i,j,u,v;

    for(i=1;i<=nodes;i++) graph[i].clear();
    for(i=1;i<=N;i++) {
        for(j=0;j<adj[i].size();j++) {
            u = color[i];
            v = color[adj[i][j]];

            if(u==v) continue;
            graph[u].pb(v);
        }
    }
    return;
}

void SCC() {
    int i,t;

    Clear(color,0);
    arr.clear();

    for(i=1;i<=N;i++) if(!color[i]) dfs1(i);
    Reverse(arr);

    Clear(color,0);
    for(i=0,t=0;i<arr.size();i++)
    if(!color[arr[i]]) dfs2(arr[i],++t);

    nodes = t;
    Rebuild_graph();
}
```

//Eulear Tour And Circuit

```
void dfs(int u) {
    int i,v;

    for(i=0;i<SZ(adj[u]);i++) {
        v = adj[u][i];
        if(v!=-1) {
            adj[u][i] = -1;
            dfs(v);
        }
    }
    order.pb(u);
}

bool possible() {
    int i,start,end,c=0;

    start = end = -1;
    for(i=0;i<nodes;i++) {
        if(indeg[i]==outdeg[i]) continue;
        else if(indeg[i]-outdeg[i]==1) {
            end = i; c++;
        }
        else if(outdeg[i]-indeg[i]==1) {
            start = i; c++;
        }
        else return false;
    }
    if(c>2) return false;

    if(start == -1) { //circuit probably
        for(i=0;i<nodes;i++)
            if(outdeg[i]) {
                start = i;break;
            }
    }
    order.clear(); //Here Finding the
    dfs(start);//Eulear tour orderings.
    Reverse(order);
    if(SZ(order)!=nodes) return false;
    //could be disconnected.
    return true;
}
```

//DFS BPM

```
bool dfs(int u) {
    if(color[u]) return false;
    color[u] = 1;

    for(int i=0;i<SZ(adj[u]);i++){
        int v = adj[u][i];
        if(par[v] == -1 || dfs(par[v]))
        {
            par[v] = u;
            return true;
        }
    }
    return false;
}
```

//HopCrftCarp BPM

```
int N,M,L[22000],R[22000],leftEle,rightEle;
vector<int>adj[22000];

bool BFS() {
    queue<int>Q;
    for(int i=1;i<=leftEle;i++) {
        if(!L[i]) {
            dist[i] = 0; Q.push(i);
        }
        else dist[i] = INF;
    }
    dist[0] = INF;
    while(!Q.empty()) {
        int u = Q.front(); Q.pop();
        if(!u) continue;
        for(int i=0;i<SZ(adj[u]);i++) {
            int v = adj[u][i];
            if(dist[R[v]] == INF) {
                dist[R[v]] = dist[u]+1;
                Q.push(R[v]);
            }
        }
    }
    return (dist[0]!=INF);
}
bool dfs(int u) {
    if(u==0) return true;

    for(int i=0;i<SZ(adj[u]);i++) {
        int v = adj[u][i];
        if(dist[u]+1 == dist[R[v]] &&dfs(R[v])) {
            L[u] = v; R[v] = u;
            return true;
        }
    }
    dist[u] = INF;
    return false;
}
int HopCraft() {
    Clear(L,0); Clear(R,0);
    int ret = 0;
    while(BFS()) {
        for(int i=1;i<=leftEle;i++)
            if(!L[i] &&dfs(i))
                ret++;
    }
    return ret;
}
```

//Topological Order….

```
void dfs(int u) {
    color[u] = 1;
    for(int i=0;i<SZ(adj[u]);i++) {
        int v = adj[u][i];
        if(!color[v]) dfs(v);
    }
    order.pb(u);
}

void TopologicalOrder(int nodes) {
    Clear(color,0); order.clear();
    for(int i=0;i<nodes;i++)
        if(!color[i]) dfs(i);
    Reverse(order);
}
```

Dinic's MaxFlow(Updated)

```
struct edges {
    int v, cap, rev;
    edges(int v=0,int cap=0,int rev=0) {
    this->v = v;this->cap = cap;this->rev = rev;
    }
};
edges edge[mMax];
vector<int>adj[nMax];
int ind,source,sink,dis[nMax];

void addEdge(int u, int v, int w, int w2) {
    edge[ind] = edges(v, w, ind+1);
    adj[u].pb(ind);
    edge[ind+1] = edges(u, w2, ind);
    adj[v].pb(ind+1);
    ind+=2;
}

int dinicBfs() {
    queue<int>Q;
    memset(dis, -1, sizeof(dis));
    Q.push(source); dis[source] = 0;

    while(!Q.empty()) {
        int u = Q.front(); Q.pop();

        for(int i=0;i<SZ(adj[u]);i++) {
            int id = adj[u][i];int v = edge[id].v;
            if(edge[id].cap && dis[v] == -1) {
                dis[v] = dis[u] + 1;
                if(v == sink) return 1;
                Q.push(v);
            }
        }
    }
    return 0;
}

int dinicDfs(int cur, int cost = INF) {
   if(cur == sink) return cost;
   int low, ans = 0;

   for(int i = 0; i<SZ(adj[cur]); i++) {
     int id = adj[cur][i];int v = edge[id].v;
     if(edge[id].cap && (dis[v] == dis[cur] + 1) &&
     (low = dinicDfs(v, min(edge[id].cap ,cost))))
     {
         edge[id].cap -= low;
         edge[edge[id].rev].cap += low;
         ans += low; cost -= low;
         if(!cost) break;
     }
   }
   return ans;
}

int dinicFlow() {
    int MaxFlow = 0;
    while(dinicBfs())
        MaxFlow += dinicDfs(source);
    return MaxFlow;
}
```

//Flow Lex. Smallest Alternate Path

```
void LexSmallFlow(parameters) {
    int i,j,u,v;
    //main code this is....
    for(i=1;i<=N;i++) {
        for(j=1;j<=M;j++) {
            u = i; v = j+N;

            if(cap[u][v]) {
                printf("0");
                cap[u][v] = cap[v][u] = 0;
            }
            else {
                cap[src][u]++;  cap[v][sink]++;
                cap[u][v] = cap[v][u] = 0;

                if(BFS(src,sink)) {
                    Update_graph(sink);
                    printf("0");
                }
                else {
                    printf("1");
                    cap[src][u]--; cap[v][sink]--;
                }
            }
        }
        printf("\n");
    }
}
```

//MinCostFlow (Using Potential)

```
struct pq {
    int node,cost;
    pq(int n,int c) {
        node = n;cost = c;
    }
    bool operator<(const pq &b)
        const{return cost>b.cost;}
};
struct edge {
    int u,v,cst,cap,rev,f;
    edge(int _u=0,int _v=0,int _cst=0,int _cap=0,
                           int _rev=0,int _f=0) {
        u = _u; v = _v; cst = _cst; cap = _cap;
        rev = _rev; f = _f;
    }
};

vector<edge>adj[2010];
int Dist[2010],potential[2010];
pii par[2010];

void addedge(int u,int v,int cst,int cap) {
    adj[u].pb(edge(u,v,cst,cap,SZ(adj[v]),0));
    adj[v].pb(edge(v,u,-cst,0,SZ(adj[u])-1,0));
}

bool Dijkstra(int source,int sink) {
    priority_queue<pq>Q;

    for(int i=0;i<=sink+1;i++) Dist[i] = INF;
    Dist[source] = 0;
    Q.push(pq(source,0));

    while(!Q.empty()) {
        int u = Q.top().node;
        int c = Q.top().cost; Q.pop();
        if(c>Dist[u]) continue;

        for(int i=0;i<SZ(adj[u]);i++) {
            edge tmp = adj[u][i];
            int v = tmp.v;
            int td = Dist[u]+tmp.cst+
                  potential[u]-potential[v];
            if(tmp.cap<=0) continue;
            if(td>=Dist[v]) continue;

            Dist[v] = td; par[v] = MP(u,i);
            Q.push(pq(v,Dist[v]));
        }
    }

    return Dist[sink]<INF;
}

void update_graph(int v,int source) {
    if(par[v].fs!=source)
        update_graph(par[v].fs,source);

    edge &tmp = adj[par[v].fs][par[v].sc];
    edge &rev = adj[tmp.v][tmp.rev];

    tmp.cap--; rev.cap++;
    tmp.f++; rev.f--;
}

int MinCostMaxFlow(int source,int sink) {
    int ans = 0;

    while(Dijkstra(source,sink)) {
        update_graph(sink,source);
        for(int i=0;i<=sink;i++)
            potential[i]+=Dist[i];
    }

    ans = 0;
    for(int i=0;i<=sink+1;i++) {
        for(int j=0;j<SZ(adj[i]);j++) {
            edge tmp = adj[i][j];
            if(tmp.f>0){
                ans+=tmp.f*tmp.cst;
            }
        }
    }

    return ans;
}
```

//Directed MST

```
#define MAX_VERTEX 100100

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;
    Clear(color,0);
    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.pb(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<SZ(Edges[v]);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].pb(e);
           }
      sort(Edges[super].begin(),Edges[super].end());

       for(j=0;j<SZ(Edges[super]);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<SZ(Edges[i]);j++) {
            adj[Edges[i][j].u].pb(Edges[i][j].v);
        }
    }
    queue<int>Q; Q.push(0);
    Clear(color,0); color[0] = 1;

    while(!Q.empty())
    {//BFS to check graph Connectivity.
        u = Q.front();Q.pop();
        for(i=0;i<SZ(adj[u]);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].pb(e);
            EdgeList.pb(e);
        }

        Clear(used,0);
        int res = isPossible();

        if(res == -1) printf("-1\n");
        else {
            Clear(used,0); Clear(color,0);
            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;
}
```

//Hungarian Algorithm

```
#define N 55

int mCost[N][N]; //main cost matrix if
            //required minimum matching
int cost[N][N]; //cost matrix
int n, max_match; //n workers and n jobs
int lx[N], ly[N]; //labels of X and Y parts
int xy[N];  //xy[x] - vertex that is matched with x,
int yx[N];  //yx[y] - vertex that is matched with y
bool S[N], T[N];//sets S and T in algorithm
int slack[N];  //as in the algorithm description
int slackx[N];//slackx[y] such a vertex, that
      //l(slackx[y])+l(y)-w(slackx[y],y)=slack[y]
int prev[N];//array for memorizing alternating paths
```

```
void init_labels() {
    memset(lx, 0, sizeof(lx));
    memset(ly, 0, sizeof(ly));
    for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++)
            lx[x] = max(lx[x], cost[x][y]);
}
void update_labels() {
    int x, y, delta = INF;
    for (y = 0; y < n; y++)
        if (!T[y]) delta = min(delta, slack[y]);
    for(x = 0; x < n; x++) if (S[x]) lx[x] -= delta;
    for(y = 0; y < n; y++) if (T[y]) ly[y] += delta;
    for(y = 0; y < n; y++)
        if(!T[y]) slack[y] -= delta;
}
void add_to_tree(int x, int prevx){

    S[x] = true; prev[x] = prevx;
    for (int y = 0; y < n; y++)
        if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
            slack[y] = lx[x] + ly[y] - cost[x][y];
            slackx[y] = x;
        }
}
void augment() {
    if (max_match == n) return;
    int x, y, root;
    int q[N], wr = 0, rd = 0;

    Clear(S,0); Clear(T,0); Clear(prev,-1);
    for (x = 0; x < n; x++)
        if (xy[x] == -1) {
            q[wr++] = root = x;
            prev[x] = -2; S[x] = true;
            break;
        }
    for (y = 0; y < n; y++) {
        slack[y] = lx[root] + ly[y] - cost[root][y];
        slackx[y] = root;
    }
    while (true) {
        while (rd < wr) {
            x = q[rd++];
            for (y = 0; y < n; y++)
                if(cost[x][y]==lx[x]+ly[y] && !T[y])
                {
                    if (yx[y] == -1) break;

                    T[y] = true;
                    q[wr++] = yx[y];
                    add_to_tree(yx[y], x);
                }
            if (y < n) break;
        }
        if (y < n) break;

        update_labels();
        wr = rd = 0;
        for (y = 0; y < n; y++)
            if (!T[y] &&  slack[y] == 0) {
                if (yx[y] == -1)
                {
                    x = slackx[y];
                    break;
                }
                else {
                    T[y] = true;
                    if (!S[yx[y]])
                    {
                       q[wr++] = yx[y];
                       add_to_tree(yx[y],slackx[y]);
                    }
                }
            }
        if (y < n) break;
    }//end of while(true).
    if (y < n)
    {
        max_match++;
        for (int cx = x, cy = y, ty; cx != -2;
                        cx = prev[cx], cy = ty)
        {
            ty = xy[cx]; yx[cy] = cx; xy[cx] = cy;
        }
        augment();
    }
}//end of augment() function

int hungarian() {
    int ret = 0; max_match = 0;
    Clear(xy,-1); Clear(yx,-1);
    init_labels(); augment();

    for (int x = 0; x < n; x++)
        ret += cost[x][xy[x]];
    return ret;
}

int main() {
    while(test--) {
        scanf("%d",&n);
        for(i=0;i<n;i++)
            for(j=0;j<n;j++)
                scanf("%d",&cost[i][j]);
        //Maximum Cost Matching....:)
        printf("Case %d: %d\n",Case++,hungarian());
    }
    return 0;
}
```

//Blossom Algorithm

```
const int MAXN = 505; // number of elements.

int n; //n no. of vertices.
vector<int> g[MAXN];
int match[MAXN]; //stores the matcings
int p[MAXN]; //array of ancestors.
int base[MAXN]; //Node numbering after compression.
int q[MAXN]; //Queue
bool used[MAXN], blossm[MAXN];

int lca (int a, int b) {
    bool used[MAXN] = { 0 };
    // From the node a climb up to the roots,
    //marking all even vertices
    for (;;) {
        a = base[a];
        used[a] = true;
        if (match[a] == -1)  break; // Got the root
        a = p[match[a]];
    }
    // Climb from node b,
    //until we find the marked vertex
    for (;;) {
        b = base[b];
        if (used[b])  return b;
        b = p[match[b]];
    }
}

void mark_path (int v, int b, int children) {
    while (base[v] != b) {
        blossm[base[v]]=blossm[base[match[v]]]=true;
        p[v] = children;
        children = match[v];
        v = p[match[v]];
    }
}

int find_path (int root) {
    Clear(used,0); Clear(p,-1);
    for (int i=0; i<n; ++i)
```

```
        base[i] = i;

    used[root] = true;
    int qh=0, qt=0;
    q[qt++] = root;
    while (qh < qt) {
        int v = q[qh++];
        for (int i=0; i<g[v].size(); ++i) {
            int to = g[v][i];
            if (base[v] == base[to]
                || match[v] == to)  continue;

            if (to == root || match[to] != -1
                && p[match[to]] != -1)
            {
                int curbase = lca (v, to);
                Clear(blossm,0);
                mark_path (v, curbase, to);
                mark_path (to, curbase, v);
                for (int i=0; i<n; ++i)
                    if (blossm[base[i]]) {
                        base[i] = curbase;
                        if (!used[i]) {
                            used[i] = true;
                            q[qt++] = i;
                        }
                    }
            }
            else if (p[to] == -1){
                p[to] = v;
                if (match[to] == -1) return to;
                to = match[to];
                used[to] = true;
                q[qt++] = to;
            }
        }
    }
    return -1;
}

int graph_match() {
    int ret = 0; Clear(match,-1);

    for (int i=0; i<n; ++i)
        if (match[i] == -1) {
            int v = find_path (i);
            if(v!=-1) ret++;
            while (v != -1) {
                int pv = p[v],  ppv = match[pv];
                match[v] = pv,  match[pv] = v;
                v = ppv;
            }
        }
    return ret;
}

int main()
{
    scanf("%d",&n);
    while(scanf("%d %d",&i,&j)==2)
    {
        i--,j--;
        g[i].pb(j);
        g[j].pb(i);
    }
    int ans = graph_match();
    printf("%d\n",ans*2);
    for(i=0;i<n;i++)
        if(match[i]>-1)
        {
            printf("%d %d\n",i+1,match[i]+1);
            match[match[i]] = -1;
        }

    return 0;
}
```

//Biconnected Component

```
vector<int>adj[MAX], nadj[MAX];
int tme[MAX], low[MAX], ID;
pii ed[2*MAX];
stack<pii>S;

void calc_bcc(int u, int v) {
    int i, j, uu, vv, cur;
    pii now; int tot=0;
    while(!S.empty()) {
        now = S.top(); S.pop();
        uu = now.first, vv = now.second;
        ed[tot++] = MP(uu, vv);
        if(u==uu && v==vv) break;
        if(u==vv && v==uu) break;
    }
    if(tot<=1) return;

    for(i=0;i<tot;i++) {
        uu = ed[i].first, vv = ed[i].second;
        nadj[uu].pb(vv); nadj[vv].pb(uu);
    }
    return;
}

void dfs_bcc(int u, int par) {
    int i, j, v;
    tme[u] = low[u] = ID++;
    for(i=0;i<SZ(adj[u]);i++) {
        v = adj[u][i];
        if(v==par) continue;
        if(tme[v]==0) {
            S.push(MP(u, v));
            dfs_bcc(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v]>=tme[u]) calc_bcc(u, v);
        }
        else if(tme[v] < tme[u]) {
            S.push(MP(u, v));
            low[u] = min(low[u], tme[v]);
        }
    }
    return;
}
```

//LCA SIMPLER

```
vi adj[MAXN];
int color[MAXN],start[MAXN],finish[MAXN],
    root,parent[MAXN][LOGMAXN],step,T,nodes;

void dfs(int u,int par){
    int i,v;
    color[u]=1; start[u]=T++;
    parent[u][0]=par;
    for(i=1;i<=step;i++)
        parent[u][i]=parent[parent[u][i-1]][i-1];
    REP(i,SZ(adj[u])) {
        v=adj[u][i];
        if(color[v]) continue;
        dfs(v,u);
    }
    finish[u]=T++;
}

//check u is anchestor of v?
bool IsAnchestor(int u,int v)
{
    if(start[u]<=start[v] && finish[u]>=finish[v])
        return true;
    return false;
}

int query_lca(int u,int v)
{
    if(IsAnchestor(u,v)) return u;
    if(IsAnchestor(v,u)) return v;

    for(int i=step;i>=0;i--)
    {
        if(!IsAnchestor(parent[u][i],v))
            u=parent[u][i];
    }
    return parent[u][0];
}
```

//HeavyLight Decomposition

```
int gene[MAXN],gene_id[MAXN],par[MAXN],
     level[MAXN],grandparent[MAXN],subtree[MAXN];
int root,nodes,id,A[MAXN],M[4*MAXN];
vi adj[MAXN];

int dfs(int u)
{
    int i,v;

    subtree[u]=1;
    REP(i,SZ(adj[u])) {
        v=adj[u][i];
        if(par[v]!=-1) continue;
        par[v]=u; level[v] = level[u]+1;
        subtree[u]+=dfs(v);
    }
    return subtree[u];
}

void prepare_Segment_tree(int u)
{
    int maxv,ind,v,i;

    gene_id[u] = id++;  maxv = ind = -1;
    REP(i,SZ(adj[u]))
    {
        v=adj[u][i];
        if(v!=par[u] && subtree[v]>maxv)
        {
            maxv=subtree[v]; ind=v;
        }
    }
    if(maxv==-1) return ;

    grandparent[ind]=grandparent[u];
    prepare_Segment_tree(ind);
    REP(i,SZ(adj[u]))
    {
        v=adj[u][i];
        if(v!=ind && v!=par[u])
        {
            grandparent[v]=v;
            prepare_Segment_tree(v);
        }
    }
}

int main() {
    while(test--) {
        scanf("%d",&nodes);
        REP(i,nodes) scanf("%d",&gene[i]);
        REP(i,nodes+1) adj[i].clear();

        REP(i,nodes-1)
        {
            scanf("%d %d",&u,&v);
            adj[u].pb(v); adj[v].pb(u);
        }
        root=0; Clear(par,-1); Clear(level,0);
        par[root]=root; dfs(root); id = 0;

        grandparent[root]=root;
        prepare_Segment_tree(root);

        REP(i,nodes)  A[gene_id[i]]=gene[i];

        Init(1,0,nodes-1);
        printf("Case %d:\n",Case++);
        scanf("%d",&Q);
        REP(q,Q)
        {
          scanf("%d",&type);
          if(type==0)
          {
            scanf("%d %d",&u,&v); ans=0;
            while(grandparent[u]!=grandparent[v])
            {
                if(level[grandparent[u]] >
                   level[grandparent[v]]) swap(u,v);
                i=gene_id[grandparent[v]];
                j=gene_id[v];
                ans+=query(1,0,nodes-1,i,j);
                v=par[grandparent[v]];
            }
            if(gene_id[u]>gene_id[v])
            swap(u,v); int xx = nodes;
            ans = ans +
              query(1,0,xx-1,gene_id[u],gene_id[v]);
            printf("%d\n",ans);
          }
          else
          {
              scanf("%d %d",&i,&val);
              update(1,0,nodes-1,gene_id[i],val);
          }
        }
    }
    return 0;
}
```

//Knuth Morris Pattern

```
void Compute_Failure_Function(string P,int par[]) {
    int i,k;
    par[0]=0; k=0; freq[0]=1;
    for(i=1;i<SZ(P);i++) {
        while(k>0 && P[k]!=P[i]) k=par[k-1];
        if(P[k]==P[i]) k++;
        if(k>0) freq[i]=freq[k-1]+1;
        else freq[i]=1;
        par[i]=k;
    }
}
```

//Suffix Automata Standard

```
#define STATE_SIZE 2*SIZE
struct state {
    int len,link,visit,cnt;
    map<char,int>child;
};
state T[STATE_SIZE];
char temp[SIZE];
int sz,last;
string str;
void Initialize(int N) {
    int i;
    REP(i,2*N) {
        T[i].len=T[i].visit=0; T[i].link=-1;
        T[i].child.clear();
    }
    sz=1; last=0;
}

void add_char(char ch) {
  int cur=sz++;
  T[cur].len=T[last].len+1; T[cur].cnt=1;
  int p;
  for(p=last;p!=-1 && T[p].child[ch]==0;p=T[p].link)
    T[p].child[ch]=cur;

  if(p==-1)  T[cur].link=0;
  else {
    int q=T[p].child[ch];

    if(T[p].len+1==T[q].len) T[cur].link=q;
    else {
      int clone=sz++;
      T[clone].link=T[q].link;
      T[clone].len=T[p].len+1;
      T[clone].child=T[q].child;
      for(p=p;p!=-1&& T[p].child[ch]==q;p=T[p].link)
        T[p].child[ch]=clone;
      T[cur].link=T[q].link=clone;
    }
  }
  last=cur;
}

vector<int>GraphByLen[SIZE];
void DP_Processing(){
    int i,j,u;
    REP(i,SZ(str)+1) GraphByLen[i].clear();
    REP(i,sz) GraphByLen[T[i].len].pb(i);

    for(i=SZ(str);i>=1;i--) {
        REP(j,SZ(GraphByLen[i])) {
            u=GraphByLen[i][j];
            T[T[u].link].cnt+=T[u].cnt;
        }
    }
}
```

//Cyclic Quest

```
char ch;
int solve(string s,int VisitNo)
{
    int L,R,cur,len=SZ(s),ans,last,l,i,p;
    s=s+s; last=l=ans=0;
    REP(i,SZ(s)) {
        ch=s[i];
        if(T[last].child[ch]) {
            last=T[last].child[ch]; l++;
            if(l==len) {
                if(T[last].visit!=VisitNo)
                {
                    ans+=T[last].cnt;
                    T[last].visit=VisitNo;
                }

                if(T[T[last].link].len==l-1)
                    last=T[last].link; l--;
            }
        }//end of .. if(T[last].child[ch])
        else {
            p=T[last].link;
            while(p!=-1 && T[p].child[ch]==0)
                p=T[p].link;
            if(p==-1) last=l=0;
            else {
                last=T[p].child[ch];
                l=T[p].len+1;
            }
        }
    }
    return ans;
}
```

//Manachar Algorithm

```
while(test--) {
    scanf("%s",str); s = "#";
    for(i=0;str[i];i++)
        {s+=str[i]; s+="#";}
    P[0] = 0; center = 0; right = 0, mxLen = -1;

    for(i=1;i<s.size();i++) {
        if(right>i) P[i]=min(P[2*center-1],right-i);
        else P[i] = 0;
        while((i-P[i]-1)>=0
              && s[i+P[i]+1]==s[i-P[i]-1])P[i]++;

        if(i+P[i]>right)
        {
            right = i+P[i]; center = i;
        }
        if(P[i]>mxLen)
            mxLen = P[i],ind = i;
    }
    result = "";
    for(i=ind-P[ind];i<=ind+P[ind];i++)
        if(s[i]!='#') result+=s[i];
    cout<<result<<endl;
}
```

//LexSmallCyclicString...

```
int lexSmallCyclicString(string str)
{
    int i=0,j=1,k=0; str = str+str;
    while(i+k<SZ(str) && j+k<SZ(str))
    {
        if(str[i+k]==str[j+k])  k++;
        else if(str[i+k]>str[j+k])
            {i = i+k+1; if(i<=j) i=j+1; k=0;}
        else
            {j = j+k+1;if(j<=i) j = i+1;k=0;}
    }
    return min(i,j);
}
```

//Histogram for Largest Rectangle

```
int Histogram_Largest_Rectangle() {
    int ans=0,s_sz=0;
    H[0]=0;  H[N+1]=0;
    for(int a=1; a<=N+1; a++) {
        while(s_sz && H[S[s_sz]]>=H[a]) {
            ans=max(ans,H[S[s_sz]]*(a-S[s_sz-1]-1));
            s_sz--;
        }
        S[++s_sz] = a;
    }
    return ans;
}
```

//Longest Common Substring(Automata)

```
int longestCommonsubstring()
{
    int last=0,len=0,i,p,j,u,v;

    Clear(maxi,0);
    REP(i,SZ(str))
    {
        char ch=str[i]-'a';
        if(T[last].child[ch])
        {
            last=T[last].child[ch];
            len++;
            maxi[last]=max(maxi[last],len);
        }
        else
        {
            p=T[last].link;
            while(p!=-1 && T[p].child[ch]==0)
                p=T[p].link;
            if(p==-1)
            {
                len=0; last=0;
            }
            else
            {
                last=p; len=T[p].len; i--;
            }
        }
    }
    for(i=L;i>0;i--)
    {
        REP(j,SZ(adj[i]))
        {
            u=adj[i][j]; v=T[u].link;
            if(maxi[u]) maxi[v]=T[v].len;
            lcs[u]=min(lcs[u],maxi[u]);
        }
    }
    return 0;
}
```

//Aho-Corasick Standard Code

```
#define STATE_SIZE 10010

struct state {
    int mask,link,child[26];
    int terminal;
};
state T[STATE_SIZE];
int sz;
vector<string>words;

void Initialize(int N) {
    int i;
    REP(i,N) {
        T[i].mask=T[i].link=0;
        Clear(T[i].child,0); T[i].terminal=-1;
    }
    sz=1;
}

void BuildAhoCorasick() {
    int i,j,last,ch,u,v,p;
    queue<int>Q;

    REP(i,SZ(words)) {
        last=0;
        REP(j,SZ(words[i])) {
            ch=words[i][j]-'a';
            if(T[last].child[ch]==0)
            T[last].child[ch]=sz++;
            last=T[last].child[ch];
        }
        T[last].terminal=i;
        T[last].mask=two(i);
    }

    REP(ch,26) {
        if(T[0].child[ch]) {
            Q.push(T[0].child[ch]);
            T[T[0].child[ch]].link=0;
        }
    }

    while(!Q.empty()) {
        u=Q.front();Q.pop();

        REP(ch,26) {
            if(T[u].child[ch]) {
                v=T[u].child[ch];
                p=T[u].link;

                while(p!=0 && T[p].child[ch]==0)
                    p=T[p].link;
                T[v].link=T[p].child[ch];
                T[v].mask|=T[T[v].link].mask;
                Q.push(v);
            }
            else
              T[u].child[ch]=T[T[u].link].child[ch];
        }
    }
}
```

//Expression Evaluation

```
string in2post(string in){
    stack<char>stk; char temp; string post;
    int i,val1,val2;

    stk.push('(');  in+=")";
    for(i=0;i<in.size();i++)
    {
        if(isalpha(in[i]) || isdigit(in[i]))
            post.push_back(in[i]);
        else if(in[i]=='(')  stk.push(in[i]);
        else if(in[i]==')')
        {
            while(true){
                temp=stk.top(); stk.pop();
                if(temp=='(')  break;
                post.push_back(temp);
            }
        }
        else if(isoperator(in[i])) {
            val1=precedence(in[i]);
            while(true)
            {
                temp=stk.top(); stk.pop();
                val2=precedence(temp);
                if(val2<val1){
                    stk.push(temp); stk.push(in[i]);
                    break;
                }
                post.push_back(temp);
            }
        }
    }
    return post;
}

bool isoperator(char ch)
{
    if(ch=='+' || ch=='-' || ch=='*' ||
    ch=='/' || ch=='^') return true;
    return false;
}

int precedence(char sign){
    if(sign=='+' || sign=='-') return 1;
    if(sign=='*' || sign=='/') return 2;
    if(sign=='^') return 3;
    return 0;
}
```

//Boolean Gauss

```
long long BoolGauss(vector <string> bo)
{
    LL gf[50];
    int s=SZ(bo),l=SZ(bo[0]),cnt=0,i,j;
    mem(gf,0);
    rii(s) rij(l)if(bo[i][j]=='Y') gf[i] |= p2(j);
    rii(l)
    {
        j = cnt;
        while(j<s && !(p2(i)&gf[j])) j++;
        if(j<s) swap(gf[j],gf[cnt]);
        else continue;
        if(p2(i)&gf[cnt]) cnt++;
        fij(cnt,s) if(p2(i)&gf[j]) gf[j]^=gf[cnt-1];
    }
    return p2(cnt);
}
```

//Gauss Elimination

```
typedef vector<double> vd;
typedef vector<vd> vvd;

const double EPS = 1e-8;
const int INF = (1<<10);
int gauss (vvd&a,vd& 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]))
                sel = i;

        if (abs (a[sel][col]) < EPS) continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        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;
    }

    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];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j) sum+=ans[j]*a[i][j];
        if (abs (sum - a[i][m]) > EPS) return 0;
    }

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

//Matrix Exponentiation

```
struct matrix
{
    int row,col;
    unsigned mat[100][100];
    matrix(int _row=0,int _col=0)
    {
        row=_row;col=_col;
        for(int i=0;i<row;i++)
            for(int j=0;j<col;j++) mat[i][j]=0;
    }
    matrix operator*(const matrix &B)const
    {
        int i,j,k;
        matrix temp(row,B.col);
        REP(i,row) REP(j,B.col) REP(k,col)
            temp.mat[i][j]=(temp.mat[i][j]+
                        mat[i][k]*B.mat[k][j]);
        return temp;
    }
    matrix operator+(const matrix &B)const
    {
        int i,j;
        matrix temp(row,col);
        REP(i,row) REP(j,col)
            temp.mat[i][j]=mat[i][j]+B.mat[i][j];
        return temp;
    }
};

matrix PowerMat(matrix A,int P){
    int i;
    matrix R(A.row,A.col);
    REP(i,R.row) R.mat[i][i]=1;
    while(P){
        if(P&1) R=(R*A);
        P>>=1;  A=(A*A);
    }
    return R;
}
```

//Suffix Array(NlogN)

```
const int S = 200010;
int cnt[S],c[19][S],nc[S],p[S],np[S],H;
char s[S];

void SuffixArray(int n,int cls=26){
    s[n]='#'; cls++; int i,m,m1;
    memset(cnt,0,sizeof(int)*cls);
    for(i=0;i<n;i++)    cnt[s[i]]++;
    for(i=1;i<cls;i++)  cnt[i]+=cnt[i-1];
    for(i=n-1;i>-1;i--) p[--cnt[s[i]]]=i;

    c[0][p[0]]=0;
    for(cls=0,i=1;i<n;i++){
        if(s[p[i]]!=s[p[i-1]]) cls++;
        c[0][p[i]]=cls;
    }
    cls++;
    for(H=0;(1<<H)<n;H++){
        for(i=0;i<n;i++){
            np[i]=p[i] - (1<<H);
            if(np[i]<0) np[i]+=n;
        }
        memset(cnt,0,sizeof(int)*cls);
        for(i=0;i<n;i++)    ++cnt[c[H][np[i]]];
        for(i=1;i<cls;i++)  cnt[i]+=cnt[i-1];
        for(i=n-1;i>-1;i--)
            p[--cnt[c[H][np[i]]]]=np[i];
        c[H+1][p[0]]=0;
        for(cls=0,i=1;i<n;i++){
            m1 = (p[i-1]+(1<<H))%n;
            m = (p[i]+(1<<H))%n;
            if(c[H][p[i]] != c[H][p[i-1]]
            || c[H][m] != c[H][m1]) cls++;
            c[H+1][p[i]]=cls;
        }
        cls++;
    }
}

int lcp(int a,int b){
    int m=0,i;
    for(i=H-1;i>-1;i--)
        if(c[i][a]==c[i][b])
        { m+=(1<<i); a+=(1<<i); b+=(1<<i); }
    return m;
}
```

//Hashing Code(2D Pattern Match)

```
struct HashMap {
    ll H1,H2;
    HashMap(ll _H1=0,ll _H2=0)
    {
        H1=_H1;H2=_H2;
    }
};
bool operator==(HashMap &a, HashMap &b) {
    return ((a.H1==b.H1) && (a.H2==b.H2));
}
ll P1=1000000007,P2=1000000009;
ll B1=257,B2=1003;

HashMap str1[SIZE],str2[SIZE],str[2*SIZE+10];
char pat[SIZE][SIZE],text[SIZE][SIZE];
ll N1,N2,M2,M1,check,Par[2*SIZE+10];

void ComputeFailureFunction(ll row) {
    ll i,len,k,r1,c1;
    len=0;
    REP(i,N2) str[len++]=str1[i];
    str[len++]=HashMap(-1,-1);
    REP(i,M2) str[len++]=str2[i];

    Par[0]=k=0;
    for(i=1;i<len;i++) {
        while(k!=0 && !(str[i]==str[k])) k=Par[k-1];
        if(str[i]==str[k]) k++;
        Par[i]=k;
        if(k==N2) {
            r1=row-N1+1; c1=i-2*N2;
            printf("(%lld,%lld)\n",r1+1,c1+1);
            check=1;
        }
    }
}

void RabinKarp() {
    ll i,j,k,tB1,tB2,tH1,tH2;
    if(N1>M1 || N2>M2) return ;
    tB1=tB2=1;
    REP(i,N1-1) {
        tB1=(tB1*B1)%P1; tB2=(tB2*B2)%P2;
    }
    REP(j,N2) {
        tH1=tH2=0;
        REP(i,N1)
        {
            tH1=(B1*tH1+pat[i][j])%P1;
            tH2=(B2*tH2+pat[i][j])%P2;
        }
        str1[j]=HashMap(tH1,tH2);
    }
    REP(j,M2) {
        tH1=tH2=0;
        REP(i,N1)
        {
            tH1=(B1*tH1+text[i][j])%P1;
            tH2=(B2*tH2+text[i][j])%P2;
        }
        str2[j]=HashMap(tH1,tH2);
    }
    for(i=N1-1;i<M1;i++) {
        ComputeFailureFunction(i);
        if(i+1==M1) continue;
        REP(j,M2)
        {
            k=i-N1+1;
            str2[j].H1=((B1*(str2[j].H1-text[k][j]
                      *tB1)+text[i+1][j])%P1+P1)%P1;
            str2[j].H2=((B2*(str2[j].H2-text[k][j]
                      *tB2)+text[i+1][j])%P2+P2)%P2;
        }
    }
}
```

//Hashing(String Matching)

```
void PreProcess()
{
    ll i,tB1,tB2;
    tB1=tB2=1;
    for(i=0;i<SIZE;i++)
    {
        inv_B1[i]=BigMod(tB1,P1-2,P1);
        inv_B2[i]=BigMod(tB2,P2-2,P2);
        tB1=(tB1*B1)%P1;  tB2=(tB2*B2)%P2;
    }
}

void GenerateString1()
{
    ll i,tB1,tB2,tH1,tH2;

    tB1=tB2=1; tH1=tH2=0;
    for(i=0;i<SZ(str1);i++)
    {
        tH1 = (tH1+str1[i]*tB1)%P1;
        tH2 = (tH2+str1[i]*tB2)%P2;
        tB1=(tB1*B1)%P1; tB2=(tB2*B2)%P2;
        hash[i]=HashMap(tH1,tH2);
    }
}

HashMap getHash(ll i,ll j)
{
    if(i==0) return hash[j];
    ll tH1=((hash[j].H1-hash[i-1].H1+P1)
            *inv_B1[i])%P1;
    ll tH2=((hash[j].H2-hash[i-1].H2+P2)
            *inv_B2[i])%P2;
    return HashMap(tH1,tH2);
}
ll Calc()
{
    ll i,tB1,tB2,tB3,tH1,tH2,tH3,cnt=1;
    HashMap H1,H2;

    tB1=tB2=1;  tH1=tH2=0;
    for(i=0;i<SZ(str2) && i<SZ(str1);i++)
    {
        tH1 = (tH1+str2[i]*tB1)%P1;
        tH2 = (tH2+str2[i]*tB2)%P2;
        tB1=(tB1*B1)%P1; tB2=(tB2*B2)%P2;
        H1=HashMap(tH1,tH2);
        H2=getHash(SZ(str1)-i-1,SZ(str1)-1);
        if(H1==H2) cnt++;
    }
    return cnt;
}
```

//Java BigInteger Code

```
import java.math.BigInteger;
public class Main {
public static void main(String[] args) {
   Scanner inp = new Scanner(System.in);
while(inp.hasNextBigInteger()){
     BigInteger b=inp.nextBigInteger(),
                    a=inp.nextBigInteger();
if(0==b.compareTo(BigInteger.valueOf(0)))break;
     a=b.multiply(a);
System.out.println(a);
  }
 }
}
```

//Rectangle Union

```
struct pq{
    int x,y1,y2,sign;
    pq(int _x=0,int _y1=0,int _y2=0,int _sign=0) {
        x=_x;y1=_y1;y2=_y2;sign=_sign;
    }
};
```

```
struct Node {
    int left,right,scal,cnt;
};
Node M[400000];
pq a[100000];
vi store;

bool comp(pq p,pq q) {
    return p.x < q.x;
}
void Initialize(int node,int left,int right)
{
    M[node].left=store[left];
    M[node].right=store[right];
    M[node].scal=M[node].cnt=0;
    if(left+1>=right) return ;

    Initialize(node*2,left,(left+right)/2);
    Initialize(node*2+1,(left+right)/2,right);
}

void Update(node, left, right, i, j, val)
{
    if(i<=M[node].left && M[node].right<=j) {
        M[node].scal+=val;
        if(M[node].scal>0) {
            M[node].cnt=(M[node].right -
                                M[node].left);
        }
        else {
            if(left+1>=right)
                M[node].cnt=0;
            else
                M[node].cnt=M[node*2].cnt +
                                M[node*2+1].cnt;
        }
        return ;
    }
    if(i>M[node].right || j<M[node].left) return ;

    Update(node*2,left,(left+right)/2,i,j,val);
    Update(node*2+1,(left+right)/2,right,i,j,val);

    if(M[node].scal==0 && left+1<right)
    M[node].cnt=M[node*2].cnt+M[node*2+1].cnt;
}
int main()
{
    scanf("%d",&N);
    cnt=0;
    store.clear();
    REP(i,N) {
        scanf("%d %d %d %d",&x1,&y1,&x2,&y2);
        store.pb(y1);
        store.pb(y2);
        a[cnt++]=pq(x1,y1,y2,+1);
        a[cnt++]=pq(x2,y1,y2,-1);
    }
    N=cnt;
    sort(&a[0],&a[N],comp);
    Sort(store);
    store.resize(unique(store.begin(),
    store.end())-store.begin());
    Total=SZ(store);
    Initialize(1,0,Total-1);
    pre=a[0].x;
    ll ans=0;
    REP(i,N)
    {
      diff=a[i].x-pre;
      ans+=(ll)diff*M[1].cnt;
      pre=a[i].x;
      Update(1,0,Total-1,a[i].y1,a[i].y2,a[i].sign);
    }
    printf("Case %d: %lld\n",Case++,ans);
}
```

//2D Segment Tree

```
void build_y(nx,lx,rx,ny,ly,ry) {
    if(ly==ry) {
        if(lx==rx)
           M[nx][ny]=mat[lx][ly];
        else
           M[nx][ny]=max(M[nx*2][ny],M[nx*2+1][ny]);
        return ;
    }
    int mid=(ly+ry)/2;
    build_y(nx,lx,rx,ny*2,ly,mid);
    build_y(nx,lx,rx,ny*2+1,mid+1,ry);
    M[nx][ny]=max(M[nx][ny*2],M[nx][ny*2+1]);
}

void build_x(int node,int lx,int rx) {
    if(lx!=rx) {
        int mid=(lx+rx)/2;
        build_x(node*2,lx,mid);
        build_x(node*2+1,mid+1,rx);
    }
    build_y(node,lx,rx,1,0,N-1);
}

int queryy(int ny,int ly,int ry,int nx,int k,int l)
{
    if(k<=ly && ry<=l) return M[nx][ny];
    if(k>ry || l<ly) return -INF;

    int mid=(ly+ry)/2;
    int p=queryy(ny*2,ly,mid,nx,k,l);
    int q=queryy(ny*2+1,mid+1,ry,nx,k,l);
    return max(p,q);
}

int queryx(nx,lx,rx,i,j,k,l)
{
    if(i<=lx&&rx<=j) return queryy(1,0,N-1,nx,k,l);
    if(i>rx || j<lx) return -INF;
    int mid=(lx+rx)/2;
    int p=queryx(nx*2,lx,mid,i,j,k,l);
    int q=queryx(nx*2+1,mid+1,rx,i,j,k,l);
    return max(p,q);
}

int main() {
    build_x(1,0,N-1);

    scanf("%d %d %d",&i,&j,&S); i--;j--;
    ans=queryx(1,0,N-1,i,i+S-1,j,j+S-1);
}
```

//LIS(NlogN)

```
int a[1200000], b[1200000];

int LIS(int n)
{
    int i,j,l,r,m=0,mid;
    b[0]=-(-1u/2);
    for(i=1;i<=n;i++) b[i]=-1u/2;

    for(i=0;i<n;i++){
        l=0;r=i+1;
        while(l<r){
            mid=(l+r)/2;
            if(b[mid]<a[i]) l=mid+1;
            else r=mid;
        }
        b[r]<?=a[i]; m>?=r;
    }
    return m;
}
```

//BatchSchedule(Mono. Que+ConvexHull)

```
int tt[SIZE],ff[SIZE],T[SIZE],F[SIZE],dp[SIZE];
vector<ll>M,B;
int pointer;

//Returns true if either line l1 or
//line l3 is always better than line l2
bool bad(int l1,int l2,int l3)
{
/*
intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
set the former greater than the latter, and
cross-multiply to eliminate division
*/
    return (B[l3]-B[l1])*(M[l1]-M[l2])<
    (B[l2]-B[l1])*(M[l1]-M[l3]);
}

void add(long long m,long long b)
{
    //First, let's add it to the end
    M.push_back(m);
    B.push_back(b);
    //If the penultimate is now made irrelevant
    //between the antepenultimate and the ultimate,
    //remove it. Repeat as many times as necessary
    while (M.size()>=3  &&
        bad(M.size()-3,M.size()-2,M.size()-1))
    {
        M.erase(M.end()-2);
        B.erase(B.end()-2);
    }
}

//Returns the minimum y-coordinate of any
//intersection between a given vertical line
//and the lower envelope
long long query(long long x)
{
    //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>M.size()) pointer=M.size()-1;

    //Any better line must be to the right, since
    //query values are non-decreasing
    while (pointer<M.size()-1 && M[pointer+1]*x +
            B[pointer+1]<M[pointer]*x+B[pointer])
        pointer++;  //MinValue Wanted...
    return M[pointer]*x+B[pointer];
}
```

//MaxSum of Max K Subsequence

```
int arr[SIZE],M[SIZE],B[SIZE],ID[SIZE],sum[SIZE];
int pointer,last;

//Returns true if l2 is better than
//line l1 when slope same
bool badMin(int l1,int l2) {
    return B[l2]<B[l1];
}
void addMin(long long m,long long b,int ind) {
    B[last]=b;
    ID[last++]=ind;    {
        B[last-2]=B[last-1];
        ID[last-2]=ID[last-1];
        last--;
    }
}
long long queryMin(long long x,int lowInd) {
    if (pointer>=last) pointer=last-1;

    while(pointer<last && ID[pointer]<lowInd)
        pointer++;
    return M[pointer]*x+B[pointer];
}
```

//String Operation

```
#define REV(i,n) for (i=n;i>=0;i--)
#define FOR(i,j,n) for(i=j;i<n;i++)

string Multiplication(string a,string b){
    int i,j,multi,carry;  string ans="0",temp;
    REV(j,SZ(b)-1){
        temp=""; carry=0;
        REV(i,SZ(a)-1){
            multi=(a[i]-'0')*(b[j]-'0')+carry;
            temp+=(multi%10+'0'); carry=multi/10;
        }
        if(carry) temp+=(carry+'0');
        Reverse(temp); temp+=string(SZ(b)-j-1,'0');
        ans=Addition(ans,temp);
    }
    ans=cut_leading_zero(ans);
    return ans;
}

string Addition(string a,string b){
    int carry=0,i; string ans;
    if(SZ(a)>SZ(b)) b=string(SZ(a)-SZ(b),'0')+b;
    if(SZ(b)>SZ(a)) a=string(SZ(b)-SZ(a),'0')+a;

    ans.resize(SZ(a));
    REV(i,SZ(a)-1){
        int sum=carry+a[i]+b[i]-96;
        ans[i]=(char)(sum%10+'0'); carry=sum/10;
    }
    if(carry)   ans.insert(0,string(1,carry+'0'));
    ans=cut_leading_zero(ans);
    return ans;
}

string Division(string a,string b){
    string mod,temp,ans="0";  int i,j;
    REP(i,SZ(a)){
        mod+=a[i]; mod=cut_leading_zero(mod);
        FOR(j,0,10){
            temp=Multiplication(b,j);
            if(compare(temp,mod)==1) break;
        }
        temp=Multiplication(b,j-1);
        mod=Subtraction(mod,temp); ans+=(j-1)+'0';
    }
    mod=cut_leading_zero(mod);
    ans=cut_leading_zero(ans);
    return ans;
}

int Div_mod(string a,int k){
  int i,sum=0;REP(i,SZ(a))sum=(sum*10+(a[i]-'0'))%k;
  return sum;
}

int compare(string a,string b){
    int i;
    a=cut_leading_zero(a);b=cut_leading_zero(b);
    if(SZ(a)>SZ(b))return 1;
    if(SZ(a)<SZ(b)) return -1;
    REP(i,SZ(a))
        if(a[i]>b[i]) return 1;
        else if(a[i]<b[i]) return -1;
    return 0;
}

string cut_leading_zero(string a){
    string s=""; int i;
    if(a[0]!='0')   return a;
    REP(i,SZ(a)-1) if(a[i]!='0')  break;
    FOR(i,i,SZ(a))  s+=a[i];
    return s;
}
```

//2D Vector

```
P MV(P a,P b) {return P(b.x-a.x,b.y-a.y);}
double CP(P a,P b) {return a.x*b.y-a.y*b.x;}
double DP(P a,P b) {return a.x*b.x+a.y*b.y;}
P VR(P a,double t) {return P(a.x*cos(t)-
                a.y*sin(t),a.x*sin(t)+a.y*cos(t));}
double Angle(P a,P b)
        {return acos(DP(a,b)/VA(a)/VA(b));}
```

//3D Plane To Point

```
bool Same_Side(P plane,Pa,Pb,Pc,P d){
    P ab = MV(a,b),np;
    np=C(ab,plane);
    return (D(np,MV(a,c))*D(np,MV(a,d)))>=0;
}

doublePlane_to_Point_Dist(P a,Pb,Pc,P d)
{
    double m=OO;
    m=min(m,Segment_to_Point_Distance(a,b,d));
    m=min(m,Segment_to_Point_Distance(a,c,d));
    m=min(m,Segment_to_Point_Distance(b,c,d));

    P plane=C(MV(a,b),MV(a,c));

    if(Same_Side(plane,a,b,c,d) &&
    Same_Side(plane,b,c,a,d) &&
    Same_Side(plane,a,c,b,d))
    {
        P ad=MV(a,d);
        doublex,y;
        y=ABS(ad);
        x=D(plane,ad)/ABS(plane);
        m = min(m,fabs(x));
    }
    return m;
}
```

//3D Segment to Segment

```
const double OO=9e10;
const double ERR=1e-6;
struct P{
    double x,y,z;
    P(double xi=0,double yi=0,double zi=0):
                        x(xi),y(yi),z(zi){}
};

P MV(P a,P b) {return P(b.x-a.x,b.y-a.y,b.z-a.z);}
double D(P a,P b){return a.x*b.x+a.y*b.y+a.z*b.z;}
P C(P a,P b) {return P(a.y*b.z-a.z*b.y,
                a.z*b.x-a.x*b.z,a.x*b.y-a.y*b.x);}
double ABS(P a) { return sqrt(a.x*a.x +
                                a.y*a.y+a.z*a.z);}
P VADD(P a,P b) {return P(a.x+b.x,a.y+b.y,a.z+b.z);}
P VSUB(P a,P b) {return P(a.x-b.x,a.y-b.y,a.z-b.z);}
P VMUL(P a,double mm) {return P(a.x*mm,
                                a.y*mm,a.z*mm);}
/**
3D Rotation
a and b suould be Unit Vector.
th :: -180<= th<=180
*/
P Rotate(P a,Pb,double th){
    double bth,tot;
    if(th>90) {
        th -=90; swap(a,b); b = Rev(b);
    }
    th = th/180.0*pi; bth=cos(th);
    th=sin(th); tot=th+bth;

    return U(P((a.x*bth+b.x*th)/tot,
        (a.y*bth+b.y*th)/tot,(a.z*bth+b.z*th)/tot));
}

P p[8];
```

```
double dis3D_Seg_to_Seg(P p1,P p2,P p3,P p4)
{
    const double SMALL_NUM = 1e-8;
    P u = MV(p1,p2);
    P v = MV(p3,p4);
    P w = MV(p3,p1);
    double a = D(u,u); double b = D(u,v);
    double c = D(v,v); double d = D(u,w);
    double e = D(v,w); double D = a*c - b*b;
    double sc, sN, sD = D;
    double tc, tN, tD = D;

    if (D < SMALL_NUM) {
        sN = 0.0; sD = 1.0; tN = e; tD = c;
    }
    else {
        sN = (b*e - c*d); tN = (a*e - b*d);
        if (sN< 0.0) {
            sN = 0.0; tN = e; tD = c;
        }
        else if (sN>sD) {
            sN = sD; tN = e + b; tD = c;
        }
    }

    if (tN< 0.0) {
        tN = 0.0;
        if (-d < 0.0)   sN = 0.0;
        else if (-d > a) sN = sD;
        else { sN = -d; sD = a; }
    }
    else if (tN>tD) {
        tN = tD;
        if ((-d + b) < 0.0) sN = 0;
        else if ((-d + b) > a) sN = sD;
        else { sN = (-d + b); sD = a; }
    }
    sc = (fabs(sN) < SMALL_NUM ? 0.0 : sN/sD);
    tc = (fabs(tN) < SMALL_NUM ? 0.0 : tN/tD);
    P dP=VADD(w, VSUB(VMUL(u,sc),VMUL(v,tc)));

    return ABS(dP);
}
```

//Centers and Line

```
typedef struct {double a,b,c;}L;
typedef struct {double x,y;}P;

L MakeLine(P p1,P p2) { return (L){ p1.y-p2.y,
                p2.x-p1.x,p1.x*p2.y-p2.x*p1.y};}

L Perpendi_Line(L l,P p) {return (L){l.b,
                        -l.a,l.a*p.y-l.b*p.x};}

double DIS(P p1,P p2){ return sqrt(sqr(p1.x-p2.x)
                                + sqr(p1.y-p2.y));}

P Intersection(L l1,L l2){
    return (P){
    (l1.b*l2.c-l2.b*l1.c)/(l1.a*l2.b-l2.a*l1.b),
    (l2.a*l1.c-l2.c*l1.a)/(l1.a*l2.b-l2.a*l1.b)
    };
}
int Number_of_IntersectionPoint(L l1,L l2){
    ///-1 means same line
    ///0 means parellal
    ///1 means there's an intersection point.

    if(l1.a == 0 && l1.b == 0 && l1.c != 0)return 0;
    if(l2.a == 0 && l2.b == 0 && l2.c != 0)return 0;
    if(l1.a == 0 && l1.b == 0) return -1;
    if(l2.a == 0 && l2.b == 0) return -1;
    if(l1.a * l2.b != l2.a * l1.b) return 1;
    if(l1.a * l2.c == l1.c * l2.a &&
            l1.b * l2.c == l1.c * l2.b) return -1;
    return 0;
}
```

```
P CircumCenter(P p1,P p2,P p3)
{
    /// When all points are not colinear
    L l1=MakeLine(p1,p2);
    L l2=MakeLine(p3,p2);
    l1=Perpendi_Line(l1,(P){(p1.x+p2.x)/2.0,
                            (p1.y+p2.y)/2.});
    l2=Perpendi_Line(l2,(P){(p3.x+p2.x)/2.0,
                            (p3.y+p2.y)/2.});
    return Intersection(l1,l2);
}
// Intersection point of three lines.
//Each line devides one angle half.
P Incenter(P p1,P p2,P p3)
{
    P re;   double a,b,c;
    a = sqrt(sqr(p2.x-p3.x)+sqr(p2.y-p3.y));
    b = sqrt(sqr(p1.x-p3.x)+sqr(p1.y-p3.y));
    c = sqrt(sqr(p2.x-p1.x)+sqr(p2.y-p1.y));
    re.x = (a*p1.x+b*p2.x+c*p3.x)/(a+b+c);
    re.y = (a*p1.y+b*p2.y+c*p3.y)/(a+b+c);
    return re;
}
```

//Circle Through Three Points

```
typedef double DD;

void CircleThroughThreePoints
        (DD x1,DD y1,DD x2,DD y2,DD x3,DD y3){
    DD A,B,C,k,g,f,H,K,Y1,Y2,X1,X2,r,c,d,e;
    A = x1*x1 + y1*y1; B = x2*x2 + y2*y2;
    C = x3*x3 + y3*y3;
    X1 = x1-x2; X2 = x2-x3;
    Y1 = y1-y2; Y2 = y2-y3;
    g = (Y2*(B-A)-Y1*(C-B))/(2*(X1*Y2-X2*Y1));
    f = (B - A - (2 * g * X1) ) / ( 2 * Y1);
    k = - A - 2 * g * x1 - 2 * f * y1;
    H = -g ;  K = -f ; r = sqrt(g*g +f*f - k);
    printf("Center(%lf, %lf)\nRadius: %lf\n",H,K,r);
}
```

//Hill Climbing

```
double x[100],y[100],z[100];
double cal(double x,doubley,double z){
    return x*x+y*y+z*z;
}
voidHullClimbing(int n) {
    int i,j,t;
    double x0,y0,z0,p,dis,mx;
    p=1;
    x0=y0=z0=0;
    for(i=0;i<20000;i++) {
        mx=-1;
        t=0;
        for(j=0;j<n;j++) {
            dis=cal(x0-x[j],y0-y[j],z0-z[j]);
            if(dis>mx)  mx=dis,t=j;
        }
        x0+=p*(x[t]-x0);
        y0+=p*(y[t]-y0);
        z0+=p*(z[t]-z0);
        p*=0.999;
    }
    printf("%.10lf %.10lf %.10lf\n",x0,y0,z0);
}
```

//Ternary Search

```
double TernarySearch(double L,double R){
    double lt,rt;
    for(int i=0;i<100;i++) {
        lt = (2*L+R)/3;
        rt = (L+2*R)/3;
        if(f(lt)<f(rt)) R=rt;
        else L=lt;
    }
    return f((L+R)/2.0);
}
```

//Closest Pair of Points

```
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);
}
```

//Point In Polygon

```
double x[100], y[100];
int n;

int point_in_poly(double xx, double yy) {
    int i, j, c=0;
    for (i = 0, j = n-1; i< n; j = i++) {
        if ( ((y[i]>yy) != (y[j]>yy)) &&
        (xx<(x[j]-x[i])*(yy-y[i])/(y[j]-y[i])+x[i]))
            c = !c;
    }
    return c;
}
```

```
int main() {
    int i, check;
    double xx, yy;
    while(scanf("%d", &n)==1) {
        for(i=0;i<n;i++) {
            scanf("%lf %lf", &x[i], &y[i]);
        }
        printf("Enter the tested point\n");

        scanf("%lf %lf", &xx, &yy);
        check = point_in_poly(xx, yy);

        if(check)  printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

//Haversine Formula(SphericalDist)

```
//lat1,lat2 lon1,lon2 all are
//in degree. Radius is R.

typedef double DD
template<class T>T sqr(T a){return a*a;}
DD pi=2*acos(0),R=6378;

DD SpR_Dist(DD lat1,DD lon1,DD lat2,DD lon2) {
    DD dlon,dlat,a,d,c;
    //convertino from degree to radian.
    lat1 *= 2.*pi/360.;lat2 *= 2.*pi/360.0;
    lon1 *= 2.*pi/360.;lon2 *= 2.*pi/360.0;
    dlon = lon2 - lon1;
    dlat = lat2 - lat1;

    a = (sqr(sin(dlat/2))) + cos(lat1) *
    cos(lat2) * sqr(sin(dlon/2));
    c = 2 * atan2(sqrt(a), sqrt(1-a));
    d = R * c;
    return d;
}
```

//Pick's Theorem

$$A = i + \frac{b}{2} - 1.$$

```
LL gcd(LL a,LL b) {
    if(!a)  return b;
    if(!b)  return a;
    return __gcd(a,b);
}

LL PiksTheorem(int n) {
    LL sum=0,I,A,B,i;
    x[n]=x[0];
    y[n]=y[0];
    for(A=B=i=0;i<n;i++) {
        A+=x[i]*y[i+1]-x[i+1]*y[i];
        B +=gcd(abs(x[i]-x[i+1]),abs(y[i]-y[i+1]));
    }
    A = abs(A);
    return  I = (A-B+2)/2;
}
```

//Center of Masses

```
int convexHull(int n) {
    j=1;
    for(i=2;i<n;i++) {
        while(j && CP(MV(a[j-1],a[j]),
                  MV(a[j-1],a[i]))<=0) j--;
        a[++j]=a[i];
    }
    return j+1;
}
```

```
pair<double,double>CenterOfMess(P p[],int n){
    doubleX,Y,area=0; inti;
    /// n = convexHull(p,n);
    /// Calculate ConvexHull;
    X = Y = 0;
    p[n].x=p[0].x;
    p[n].y=p[0].y;

    for(i=0;i<n;i++)
        area+=(p[i].x*p[i+1].y-p[i+1].x*p[i].y);
    area = fabs(area);
    area*=3;
    for(i=0;i<n;i++){
        X+=(p[i].x+p[i+1].x) * (p[i].x*p[i+1].y -
                                p[i+1].x*p[i].y ) ;
        Y+=(p[i].y+p[i+1].y) * (p[i].x*p[i+1].y -
                                p[i+1].x*p[i].y ) ;
    }
    X /=area;   Y/=area;
    return make_pair(X,Y);
}
```

//Smallest Enclosing Rectangle

```
#define ERR         1e-5
#define PRE         1e-8
#define X           first
#define Y           second
#define MP          make_pair

const double eps = 1e-9;
template<class TT>TT sqr(TT a){return a*a;}
template<class TT>TT abs(TT a)
                {if(a<0)return -a; return a;}
template<class ZZ>ZZ max(ZZ a,ZZb,ZZ c)
                    {return max(a,max(b,c));}
template<class ZZ>ZZ min(ZZ a,ZZb,ZZ c)
                    {return min(a,min(b,c));}

struct P{
    double x,y;
    P(double xx=0,double yy=0){x=xx,y=yy;}

    void Rotate() {
        double xi=x,yi=y;
        y = xi; x = -yi;
    }
};
P MV(P a,P b){return P(b.x-a.x,b.y-a.y);}
double CP(P a,P b){return a.x*b.y-a.y*b.x;}
double DP(P a,P b){return a.x*b.x+a.y*b.y;}
double ABS(P a){return sqrt(DP(a,a));}
const int S = 100101;
P pvt,p[S];

bool operator<(const P &a,const P &b){
    double c = CP(MV(pvt,a),MV(pvt,b));
    if(fabs(c)>eps)   return c>0;
    return DP(MV(pvt,a),MV(pvt,a)) <
                DP(MV(pvt,b),MV(pvt,b));
}

P operator - (const P &a,const P &b)
            {return P(b.x-a.x,b.y-a.y);}

int Convexhudouble(P a[],int n) {
    int i,j; pvt=a[0];
    for(i=1;i<n;i++)
    if((pvt.y>a[i].y)||(pvt.y==a[i].y &&
                        pvt.x>a[i].x))pvt=a[i];
    sort(a,a+n);
    for(i=2,j=1;i<n;i++) {
        while(j && CP(MV(a[j-1],a[j]),
                      MV(a[j-1],a[i]))<=0) j--;
        a[++j]=a[i];
    }
    return j+1;
}
```

```
typedef pair<double,double>pdd;

pdd Rotating_Cleaper(P po[],int n)
{
    double h,w,a,b,area=1e15,per=1e15;
    P tmp;
    int i,j,k,l,ii;

    for(i=j=k=l=0;i<n;i++)
    {
        ii = (i+1)%n;
        tmp = MV(po[i],po[ii]);///L
        for(;DP(tmp,po[j]-po[(j+1)%n])>0;j=(j+1)%n)
            ;
        if((j-i+n)%n>(k-i+n)%n) k=j;
        tmp.Rotate();///U
        for(;DP(tmp,po[k]-po[(k+1)%n])>0;k=(k+1)%n)
            ;
        if((k-i+n)%n>(l-i+n)%n) l=k;
        tmp.Rotate();///R
        for(;DP(tmp,po[l]-po[(l+1)%n])>0;l=(l+1)%n)
            ;
        tmp.Rotate();
        /**
        i = Low Point,j = Left Point.
        k = Top Point,l = Right Point.
        */
        h = fabs(DP(tmp,po[k]-po[i])/ABS(tmp));
        tmp.Rotate();
        w = fabs(DP(tmp,po[j]-po[l])/ABS(tmp));
        area = min(area,h*w);
        per = min(per,h+w);
    }

    return MP(area,2*per);
}

int main() {
    int i,j,k,l,tks,ks=1,n,m;
    pair<double ,double>re;

    while(scanf("%d",&n)==1&&n) {
        for(i=0;i<n;i++)
            scanf("%lf%lf",&p[i].x,&p[i].y);
        n = Convexhudouble(p,n);
        re = Rotating_Cleaper(p,n);
        printf("%.2lf %.2lf\n",re.X,re.Y);
    }
    return 0;
}
```

//SmallestEncloseCircle(Random.Algo)

```
typedef double DD;
const double TOLL=1e-9;
struct PO{
    DD x,y;
    PO(DD xx=0,DD yy=0)
    {x=xx;y=yy;}
};
DD dist(PO a,PO b) { return sqrt((a.x-b.x) *
                (a.x-b.x)+(a.y-b.y)*(a.y-b.y) );}
bool operator<(const PO&a,const PO& b){
    if(fabs(a.x-b.x)>TOLL)return a.x+TOLL<b.x;
    return a.y+TOLL<b.y;
}
struct line{
    DD x,y, A,B,C;
    line(PO a,PO b){
        x=b.x-a.x, y=b.y-a.y;
        A=b.y-a.y; B=a.x-b.x; C=A*a.x+B*a.y;
    }
    line(){/*EMPTY*/}
};
DD cross(line a, line b) {return a.x*b.y-b.x*a.y;}
DD cross(PO a, PO b)  {return a.x*b.y-b.x*a.y;}
DD dot(line a,line b) {return a.x*b.x+a.y*b.y;}
PO line_intersection(line ab,line cd){
    DD x=-1e100,y=x;
    DD det = ab.A*cd.B - cd.A*ab.B;
    if(det == 0){
        //Lines are parallel
    }
    else{
        x = (cd.B*ab.C - ab.B*cd.C)/det;
        y = (ab.A*cd.C - cd.A*ab.C)/det;
    }
    return PO(x,y);
}
line perpendicular(line a, PO ref){
    line rv; rv.A=-a.B; rv.B=a.A;
    rv.C=rv.A*ref.x+rv.B*ref.y; return rv;
}
struct circle{
    PO cen; DD radius;
    circle(PO cc,DD r) {cen=cc; radius=r;}
};
circle from2POs(PO a,PO b){
    PO cen; cen.x=(a.x+b.x)/2;
    cen.y=(a.y+b.y)/2; DD rad=dist(cen, a);
    return circle(cen,rad);
}
circle from3points(PO a,POb,PO c){
    PO midab;
    midab.x=(a.x+b.x)/2; midab.y=(a.y+b.y)/2;
    PO midbc;
    midbc.x=(b.x+c.x)/2; midbc.y=(b.y+c.y)/2;
    lineab(a,b), bc(b,c);
    if(fabs(cross(ab,bc) )<TOLL) {
        vector<PO>tmp; tmp.push_back(a);
        tmp.push_back(b); tmp.push_back(c);
        sort(tmp.begin(),tmp.end());
        return from2POs(tmp[0],tmp[2]);
    }
    line first=perpendicular(ab,midab),
    second=perpendicular(bc,midbc);
    PO cen=line_intersection(first,second);
    DD r=dist(cen,a);
    circle rv(cen,r);
    return rv;
}
bool insidecircle(circle g,PO t){
    DD d=dist(g.cen,t);
    if(d>g.radius+TOLL) return false;
    return true;
}
circle smallestEncloseCircle(vector<PO>& p) {
    random_shuffle(p.begin(),p.end());
    circle rv=from2POs(p[0],p[1]);
    for(int i=2;i<p.size();i++)
    if(!insidecircle(rv,p[i])) {
        rv=from2POs(p[0],p[i]);
        for(int j=1;j<i;j++)
            if(!insidecircle(rv,p[j])) {
                rv=from2POs(p[j],p[i]);
                for(int k=0;k<j;k++)
                    if(!insidecircle(rv,p[k]))
                        rv=from3POs(p[i],p[j],p[k]);
            }
    }
    return rv;
}
vector<PO> v;
int main() {
    while(t--) {
        int n; scanf("%d",&n); v.resize(n);
        for(int i=0;i<n;i++)
            scanf("%lf %lf",&v[i].x,&v[i].y);
        circle rv=smallestEncloseCircle(v);
        printf("%.2lf\n%.2lf %.2lf\n",
               rv.radius,rv.cen.x,rv.cen.y);
    }
    return 0;
}
```

//Seive Yarin

```
#define MAXS 100000000
#define MAXSH (MAXS/2)
#define MAXSQ 5000
#define isprime(n) (a[n >> 4] & (1<<(((n)>>1)&7)))
//works when n is odd
char a[MAXS/16+2];
#define PN 5762000
int prime[PN],c;
void seive(){
    int i,j; memset(a,255,sizeof(a));
    a[0]=0xFE;
    for(i=1;i<MAXSQ;i++)
        if (a[i>>3]&(1<<(i&7)))
            for(j=i+i+i+1;j<MAXSH;j+=i+i+1)
                a[j>>3]&=~(1<<(j&7));
    prime[c++]=2;
    for(i=3;i<MAXS;i+=2)
        if(isprime(i))
            prime[c++]=i;
    printf("Total prime:%d\n",c);
}
```

//Extended Euclidean

```
int Extended_Euclidean(int a,int b,int &x,int &y){
    if(b==0) { x=1;y=0; return a; }
    int d=Extended_Euclidean(b,a%b,y,x);
    y=y-(a/b)*x; return d;
}
```

//Chinese Remainder Theorem

```
long long p[100],r[100];
while(test--){
    scanf("%lld",&n);
    for(i=0;i<n;i++) scanf("%lld %lld",&p[i],&r[i]);
    N=1;
    for(i=0;i<n;i++) N*=p[i];
    for(x = 0, i=0;i<n;i++){
        b=Inverse_Modulo(N/p[i],p[i]);
        x=(x+r[i]*b*(N/p[i]))%N;
    }
    x%=N;
    printf("Case %lld: %lld\n",Case++,x);
}
```

//Shanks Baby Step Giant Algo.

```
LL Shanks_Baby_step_Giant_Algo(LL a,LL b,LL m){
    /// a^x = b (%p).
    LL n = (LL)sqrt(m)+1; LL cur,an,i,re,j;
    an=BigMod(a,n,m); map<LL,LL>mp;
    for(i=1,cur=an;i<=n;i++){
        if(!mp.count(cur)) mp[cur]=i;
        cur = (cur*an)%m;
    }
    for(re = m, j=0,cur=b;j<n;j++){
        if(mp.count(cur)){
            i = n*mp[cur]-j;
            if(i<m)   re = min(re,i);
        }
        cur = (cur*a)%m;
    }
    if(re<m)    return re;
    return -1;  ///If not possible.
}
```

//Harmonic Number

```
double a[20902];
for(S = 0,i=1;i<=20900;i++) {S+=1./i; a[i] = S;}

if(n>20900)
    t = log(n) + 1/(2.*n) + 0.57721566490153286060;
else    t = a[n];
printf("Case %d: %.10lf\n",ks,t);
```

//Finding Lattice Point

```
scanf("%lld %lld %lld %lld", &x1, &y1, &x2, &y2);
dx = _abs(x1 - x2); dy = _abs(y1 - y2);
printf("Case %d: %lld\n", cs, gcd(dx, dy) + 1);
```

//How Many Divisor's of a number

$N=p_1^{q1} \times p_2^{q2} \times p_3^{q3} \times \ldots\ldots\ldots \times p_k^{qk}$

$D(N) = (q1+1) \times (q2+1) \times (q3+1) \times \ldots \times (qk+1)$

//Sum of Divisors

$$n = p_1^{e_1} * p_2^{e_2} * \cdots * p_k^{e_k}$$

$$\sigma(n) = \frac{p_1^{e_1+1}-1}{p_1-1} * \frac{p_2^{e_2+1}-1}{p_2-1} * \cdots * \frac{p_k^{e_k+1}-1}{p_k-1}$$

Eulear Totient Function

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k}$$

$$\phi(n) = \phi(p_1^{a_1}) \cdot \phi(p_2^{a_2}) \cdot \ldots \cdot \phi(p_k^{a_k}) =$$

$$= \left(p_1^{a_1} - p_1^{a_1-1}\right) \cdot \left(p_2^{a_2} - p_2^{a_2-1}\right) \cdot \ldots \cdot \left(p_k^{a_k} - p_k^{a_k-1}\right) =$$

$$= n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdot \ldots \cdot \left(1 - \frac{1}{p_k}\right).$$

//Fermat's Little Theorem

$$a^{p-1} \equiv 1 \pmod{p}$$

//Factorial Factors

$$n! = \prod_{\substack{p \le n \\ p \text{ prime}}} p^{\left[\sum_{k>0} \text{IntegerPart}\left(\frac{n}{p^k}\right)\right]}$$

//Last Non Zero digit of factorial

```
L(N)=(N!)%10
L(0)=1, L(1)=1, L(2)=2, L(3)=6, L(4)=4
L(N)= (2N/5 x  L(N/5)  x L(N%5) )%10
    = ((2N/5 %10) x  L(N/5)  x L(N%5) )%10
```

//Inverse Modulo

$$ax + by = \gcd(a, b)$$
$$ax - qm = 1,$$

//Dearrangement Recurrence:

```
if(n==1) return 0;
if(n==2) return 1;
ret+=(n-1)*rec(n-2)+(n-1)*rec(n-1);
```

//Catalan Number

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \quad \text{for } n \ge 0;$$

//Stirling Number of first Kind

$$S(m,n)=S(m-1,n-1)-(m-1)S(m-1,n)$$

//Stirling Number of second Kind

$$S(m,n)=S(m-1,n-1)+nS(m-1,n)$$

Number of Trailing Zero's of N! Base B

(1) $Z_p(n)=\sum_{i\geq 1}\left\lfloor \frac{n}{p^i}\right\rfloor = \frac{n-\sigma_p(n)}{p-1}$, where $\sigma_p(n)$ is the sum of the digits of the base p expansion of n.

(2) $Z_{p^r}(n)=\left\lfloor \frac{Z_p(n)}{r}\right\rfloor$ for every $r\geq 1$.

(3) If $b=p_1^{r_1}\cdots p_s^{r_s}$, then $Z_b(n)=\min_{1\leq i\leq s} Z_{p_i^{r_i}}(n)$.

//Liner Diophantine Equation

(i) The linear Diophantine equation

$$ax+by=c$$

has a solution if and only if $d=\gcd(a,b)$ divides c.

(ii) If $d \mid c$, then one solution may be found by determining u and v su that $d=ua+vb$ and then setting

$$x_0=uc/d \quad \text{and} \quad y_0=vc/d.$$

All other solutions are given by

$$x=x_0+(b/d)t, \qquad y=y_0-(a/d)t$$

for $t\in\mathbb{Z}$.

//Solution

```
n1m1 + n2m2 = n -----(1)
Minimize -> c1m1 + c2m2
g = gcd(n1, n2)
n1m1' + n2m2' = g ---- (2)
Multiplying by (n/g) ->
n1m1'' + n2m2'' = n --- (3)
From (1) and (3) ->
m1 = m1'' + (n2/g) t
m2 = m2'' - (n1/g) t
Here, t is an integer parameter.
From the conditions -> m1≥0, m2≥0, n1>0, n2>0,
ceil(-m1'' g/n2) ≤ t ≤ floor(m2'' g/n1)

c = c1m1 + c2m2
  = c1m1'' +c2m2'' + (c1n2/g - c2n1/g) t
  = a + bt
As this is a linear function, its minimum value
will be on either of the boundaries.
```

//Binomial Coefficient Info's

	0	1	2	3	4	5	6	7	8
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	
8	1	8	28	56	70	56	28	8	1

4.1.2 Basic Binomial Coefficient Identities

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad \text{Pascal Rec} \quad (4.1.1)$$

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!} \quad \text{Falling Power Formula} \quad (4.1.2a)$$

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!} \quad \text{Factorial Formula} \quad (4.1.2b)$$

$$\binom{n}{k} = \binom{n}{n-k} \quad \text{Symmetry} \quad (4.1.3)$$

$$\sum_{k=0}^{n}\binom{n}{k} = 2^n \quad \text{Row-Sum} \quad (4.1.4)$$

$$\sum_{r=0}^{n}\binom{r}{c} = \binom{n+1}{c+1} \quad \text{Column-Sum} \quad (4.1.5)$$

$$\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n} \quad \text{SE Diagonal} \quad (4.1.6a)$$

$$\sum_{k=0}^{m}\binom{n-k}{m-k} = \binom{n+1}{m} \quad \text{NW Diagonal} \quad (4.1.6b)$$

$$\sum_{k=0}^{n}\binom{n-k}{k} = f_{n+1} \quad \text{Fibonacci NE Diagonal} \quad (4.1.7)$$

$$\binom{n}{k}k = n\binom{n-1}{k-1} \quad \text{Absorption} \quad (4.1.8)$$

$$\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k} \quad \text{Subset-of-a-Subset} \quad (4.1.9)$$

$$\sum_{j=0}^{n}\binom{n}{j}\binom{m}{k-j} = \binom{n+m}{k} \quad \text{Vandermonde Convo}$$

Theorem 4.1.10. *Let n and k be non-negative integers. Then*

$$\binom{n}{k} \equiv \begin{cases} 0 \bmod 2 & \text{if } n \text{ is even and } k \text{ is odd} \\ \binom{\lfloor n/2\rfloor}{\lfloor k/2\rfloor} \bmod 2 & \text{otherwise} \end{cases}$$

//Equation of Sums

Sums

$$\sum_{k=0}^{n} k = n(n+1)/2 \qquad \sum_{k=a}^{b} k = (a+b)(b-a+1)/2$$

$$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6 \qquad \sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$$

$$\sum_{k=0}^{n} k^4 = (6n^5+15n^4+10n^3-n)/30 \qquad \sum_{k=0}^{n} k^5 = (2n^6+6n^5+5n^4-n^2)/12$$

$$\sum_{k=0}^{n} x^k = (x^{n+1}-1)/(x-1) \qquad \sum_{k=0}^{n} kx^k = (x-(n+1)x^{n+1}+nx^{n+2})/(x-1)^2$$

$$1+x+x^2+\cdots = 1/(1-x)$$

//Trigonometric Equations

Trigonometric identities

$\sin(\alpha+\beta)=\sin\alpha\cos\beta+\cos\alpha\sin\beta$ $\cos(\alpha+\beta)=\cos\alpha\cos\beta-\sin\alpha\sin\beta$

$\sin(\alpha-\beta)=\sin\alpha\cos\beta-\cos\alpha\sin\beta$ $\cos(\alpha-\beta)=\cos\alpha\cos\beta+\sin\alpha\sin\beta$

$\tan(\alpha+\beta)=\frac{\tan\alpha+\tan\beta}{1-\tan\alpha\tan\beta}$ $\sin 2\alpha=2\sin\alpha\cos\alpha$, $\cos 2\alpha=\cos^2\alpha-\sin^2\alpha$

$\cos^2\alpha=\frac{1}{2}(1+\cos 2\alpha)$ $\sin^2\alpha=\frac{1}{2}(1-\cos 2\alpha)$

$\sin\alpha+\sin\beta=2\sin\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$ $\cos\alpha+\cos\beta=2\cos\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$

$\sin\alpha-\sin\beta=2\sin\frac{\alpha-\beta}{2}\cos\frac{\alpha+\beta}{2}$ $\cos\alpha-\cos\beta=-2\sin\frac{\alpha+\beta}{2}\sin\frac{\alpha-\beta}{2}$

$\tan\alpha+\tan\beta=\frac{\sin(\alpha+\beta)}{\cos\alpha\cos\beta}$ $\cot\alpha+\cot\beta=\frac{\sin(\alpha+\beta)}{\sin\alpha\sin\beta}$

$\sin\alpha\sin\beta=\frac{1}{2}[\cos(\alpha-\beta)-\cos(\alpha+\beta)]$ $\cos\alpha\cos\beta=\frac{1}{2}[\cos(\alpha-\beta)+\cos(\alpha+\beta)]$

$\sin\alpha\cos\beta=\frac{1}{2}[\sin(\alpha+\beta)+\sin(\alpha-beta)]$ $\sin' x=\cos x$, $\cos' x=-\sin x$

Law of sines: $\frac{a}{\sin A}=\frac{b}{\sin B}=\frac{c}{\sin C}=2R_{out}$. Inscribed/outscribed circles: $R_{out}=\frac{abc}{4S}$, $R_{in}=\frac{2S}{a+b+c}$

Law of cosines: $c^2=a^2+b^2-2ab\cos C$. Heron: $\sqrt{s(s-a)(s-b)(s-c)}$, $s=\frac{a+b+c}{2}$.

Law of tangents: $\frac{a+b}{a-b}=\frac{\tan[\frac{1}{2}(A+B)]}{\tan[\frac{1}{2}(A-B)]}$ Δ's area, given side and adjacent angles: $\frac{c^2}{2(\cot\alpha+\cot\beta)}$