**Team Bozonghereh Notebook**

**Table of content**

[**Graph**](#_kdsdhhj876z0) **1**

[2-SAT(SCC-Topological Sort)](#_iusjh5xn36ec) 1

[Topological Sort(Second Kind)](#_r5zf8rv6eyix) 1

[Bellman-Ford](#_vk61zpkrskdi) 2

[Floyd-Warshall](#_5nrlcmkrm0ps) 2

[DSU](#_3xf9d573ukz2) 2

[Flow (Edmonds-Karp)](#_kvf7b7dmabzg) 2

[Flow (Dinic)](#_1aj3tsyq9st3) 3

[Flow(Push-Relabel)](#_70vjyl3wqkuz) 4

[Min Cost Max Flow](#_en52trukww8w) 5

[Bipartite Matching](#_gn904663sd69) 6

[Weighted Bipartite Matching](#_e3p5uv81bgyp) 6

[Cut Vertex(Biconnected Component)](#_jl91gvrozp7g) 7

[Cut Edge](#_l2e273s1g88) 8

[LCA](#_25y86450uv7y) 8

[Eulerian Tour](#_n76hh0tijtvv) 9

[**Data Structures**](#_m71pqzfrr7au) **9**

[Ordered Set](#_aifhy89c2k0l) 9

[Heavy Light](#_ihl0hn8z9tw8) 9

[**Math**](#_l9wvyck8kss9) **11**

[Power Lemma](#_67gkjqczn62y) 11

[Wilson](#_u6tuac26jbbr) 11

[AKS](#_q0p4ynvkqmpq) 11

[Newton’s Method](#_xk7xoqbm9n2u) 11

[Extended Euclid- Chinese Remainder](#_6eidzvj32d38) 11

[Miller](#_n6ihej8lv6vv) 11

[Linear System](#_vcb0jm62qthx) 12

[Catalan Number](#_kc54fhi17ctq) 13

[FFT](#_fjrxlzmjhh96) 13

[**Geometry**](#_c2kchnrogu78) **14**

[Simplex Algorithm](#_x7t8xneyq2n9) 14

[Closest Pair](#_37c4mnoww2w1) 15

[Areas and Angles](#_wx4wq0bo450b) 16

[Lines Points Intersect](#_nqosux73cziy) 17

[Centers and Centroids](#_6k4vvfn3aa41) 18

[Point in Polygon](#_i6irv9xjuqzu) 18

[Big Intersections](#_78cdpbbkk3k0) 19

[Convex Hull](#_3nm2hzvhswt6) 20

[Misc](#_8sgalqi0de5p) 21

[**Dynamic Programming**](#_6am3sd77l5t) **21**

[LIS](#_dc8tcmiem61h) 21

[Matrix DP](#_7sgs0yv3i89n) 21

[**String**](#_9mz4b1a345v1) **21**

[Hash](#_9y28gu13c4pe) 21

[KMP](#_hznr5fmlgbkr) 22

[**Misc**](#_erk4t3rg25di) **22**

[Ternary Search](#_7e8e8ycl3rhg) 22

[Grid](#_f3bahe9v5ocr) 22

# 

# 

# Graph

#### 2-SAT(SCC-Topological Sort)

//O(E) 1-base

//if you want use SCC only ignore

//last method booleans are 1 to n / ~i = i+n

//add edge with inverse! a or b == ~b -> a == == a -> ~b

int n,m,cnt = 1,verticesScc[MAX\_N];

vector<int>g[MAX\_N],g1[MAX\_N], tSort, scc[MAX\_N];

bool mark[MAX\_N],ans[MAX\_N],satisfy = true;

void topologicalSort(int v) {

mark[v] = true;

for(int u:g[v])

if(!mark[u])

topologicalSort(u);

tSort.push\_back(v);

}

void dfs(int v) {

mark[v] = true;

scc[cnt].push\_back(v);

verticesScc[v] = cnt;

for(int u:g1[v])

if(!mark[u])

dfs(u);

}

void SCC() {

for(int i = 1 ; i <= n ; i++)

if(!mark[i])

topologicalSort(i);

reverse(tSort.begin(), tSort.end());

memset(mark, false, sizeof mark);

for(int v:tSort)

if(!mark[v]) {

cnt++;

dfs(v);

}

cnt--;

}

int inverse(int t) {

if(t <= n)

return t+n;

return t-n;

}

bool check() {

for(int i = 1 ; i<= n ; i++)

if(verticesScc[i] == verticesScc[i+n])

return false;

return true;

}

//one of the expressions are (x or y)

void addExpresion(int x,int y) {

g[inverse(x)].push\_back(y);

g[inverse(y)].push\_back(x);

g1[y].push\_back(inverse(x));

g1[x].push\_back(inverse(y));

}

void sat() {

n \*= 2;

SCC();

n/=2;

if(!check()) {

satisfy = false;

return;

}

memset(mark, false, sizeof mark);

reverse(tSort.begin(),tSort.end());

for(int v:tSort) {

if(v <= n && !mark[v]) {

mark[v] = true;

ans[v] = true;

}

if(v > n && !mark[v-n])

mark[v-n] = true;

}

}

#### Topological Sort(Second Kind)

//O(E log n) 1-base

for(int i = 1 ; i <= n ; i++)

if(deg[i] == 0)

s.insert(i);

while(s.size() > 0) {

int v = \*s.begin();

s.erase(s.begin());

tSort.push\_back(v);

for(int u:g1[v]) {

deg[u]--;

if(deg[u] == 0)

s.insert(u);

}

}

#### Bellman-Ford

//O(VE) 1-base

//Directional-we can solve inequalities like : x(i) - x(j) <= c we make //one node for each x(i) we make a source with edge weight 0 to all other //nodes we put a edge from j to i with weight c bellman ford from source //x[i] is dis[i]

ll n,m,dis[MAX\_N];

//first Node, second Node, weight

vector<pair<pair<ll,ll>,ll > > edges;

bool negativeCycle;

void bellmanFord() {

memset(dis, 31, sizeof dis);

dis[1] = 0;

for(int i = 1 ; i <= n ; i++)

for(pair<pair<ll,ll>,ll > e:edges)

if(dis[e.first.first] +e.second < dis[e.first.second])

dis[e.first.second] = dis[e.first.first]+e.second;

for(pair<pair<ll,ll>,ll > e:edges)

if(dis[e.first.first] +e.second < dis[e.first.second])

negativeCycle = true;

}

#### Floyd-Warshall

//O(V^3) 1-base

int n,m,dis[MAX\_N][MAX\_N],g[MAX\_N][MAX\_N];

//directional, g[i][j] = INF

void floydWarshall() {

for(int i = 0 ; i < MAX\_N ; i++)

for(int j = 0 ; j < MAX\_N ; j++)

dis[i][j] = min(INF,g[i][j]);

for(int i = 1 ; i <= n ; i++)

dis[i][i] = 0;

for(int k = 1 ; k <= n ; k++)

for(int i = 1 ; i <= n ; i++)

for(int j = 1 ; j <= n ; j++)

if(dis[i][j] > dis[i][k] + dis[k][j])

dis[i][j] = dis[i][k] + dis[k][j];

//for using minimax and maximin minimax :

//dis[i][j] = min(dis[i][j],max(dis[i][k],dis[k][j]))

//dis[i][j] + dis[j][i] < 0 -> negative cycle

}

#### DSU

//O(log(n)) no-base

int n, m, par[MAX\_N], sz[MAX\_N];

void create(int v) {

par[v] = v;

sz[v] = 1;

}

int find\_par(int v) {

if(par[v] == v)

return v;

par[v] = par[par[v]];

return find\_par(par[par[v]]);

}

void join(int v, int u) {

u = find\_par(u);

v = find\_par(v);

if(u == v)

return;

if(sz[v] < sz[u])

swap(u, v);

par[u] = v;

sz[v] += sz[u];

}

#### Flow (Edmonds-Karp)

//O((min(f|E|, |V||E|^2)) 1-base

int n,m,s,t,c[MAX\_N][MAX\_N],cf[MAX\_N][MAX\_N],par[MAX\_N],maxFlow;

bool mark[MAX\_N];

queue<int>q;

bool bfs() {

memset(mark, false, sizeof mark);

while(q.size()) q.pop();

q.push(s);

mark[s] = true;

par[s] = -1;

while(q.size() > 0) {

int a = q.front();

q.pop();

for(int i = 1 ; i <= n ; i++) {

if(!mark[i] && cf[a][i] > 0) {

par[i] = a;

mark[i] = true;

q.push(i);

}

}

}

return mark[t];

}

void edmondsKarp() {

while(bfs()) {

vector<int>path;

int tmp = t;

while(tmp != -1) {

path.push\_back(tmp);

tmp = par[tmp];

}

reverse(path.begin(),path.end());

int MIN = INF;

for(int i = 0 ; i < path.size() -1 ; i++) {

int a = path[i];

int b = path[i+1];

MIN = min(MIN,cf[a][b]);

}

for(int i = 0 ; i < path.size() -1 ; i++) {

int a = path[i];

int b = path[i+1];

cf[a][b] -= MIN;

cf[b][a] += MIN;

}

maxFlow += MIN;

}

}

void addDirectionalEdge(int u, int v, int cap) {

c[u][v] += cap, cf[u][v] += cap;

}

void addBiDirectionalEdge(int u, int v, int cap) {

c[u][v] += cap, c[v][u] += cap, cf[u][v] += cap, cf[v][u] += cap;

}

#### Flow (Dinic)

//O(V^2\*E) no-base  
struct Edge {  
 int to,reverseIndex,cap,flow;  
};  
int n,m,s,t,maxFlow,dis[MAX\_N];  
vector<Edge>g[MAX\_N];  
queue<int>q;  
void addEdge(int u, int v, int cap) {  
 Edge x,y;  
 x.to = v, y.to = u;  
 x.cap = cap, y. cap = 0;  
 x.flow = y.flow = 0;  
 x.reverseIndex = g[v].size();  
 y.reverseIndex = g[u].size();  
 g[u].push\_back(x);  
 g[v].push\_back(y);  
}  
bool bfs() {  
 memset(dis, 31, sizeof dis)  
 while(q.size()) q.pop();  
 q.push(s);  
 dis[s] = 0;  
 while(q.size() > 0) {  
 int v = q.front();  
 q.pop();  
 for(Edge x:g[v]) {  
 int u = x.to;  
 if(dis[u] == INF && x.flow < x.cap) {  
 dis[u] = dis[v]+1;  
 q.push(u);  
 }  
 }  
 }  
 return (dis[t] != INF);  
}  
int dfs(int v,int f) {  
 if(v == t)  
 return f;  
 for(int i = 0 ; i < g[v].size() ; i++) {  
 Edge &x = g[v][i];  
 int u = x.to;  
 if(x.cap <= x.flow) continue;  
 if(dis[u] == dis[v]+1) {  
 int tmp = dfs(u,min(f,x.cap-x.flow));  
 if(tmp > 0) {  
 x.flow += tmp;  
 g[u][x.reverseIndex].flow -= tmp;  
 return tmp;  
 }  
 }  
 }  
 return 0;  
}  
void dinic() {  
 while(bfs())   
 while(int tmp = dfs(s,INF))  
 maxFlow += tmp;  
}

#### Flow(Push-Relabel)

//O(V^3) 0-base

struct Edge {

int from, to, cap, flow, index;

Edge(int from, int to, int cap, int flow, int index) :

from(from), to(to), cap(cap), flow(flow), index(index) {}

};

int n, m, s, t, maxFlow, dist[MAX\_N], active[MAX\_N], CNT[2\*MAX\_N];

vector<Edge> g[MAX\_N];

ll excess[MAX\_N];

queue<int> q;

void CLEAR() {

maxFlow = 0;

while(q.size()) q.pop();

memset(CNT, 0, sizeof CNT);

for(int i = 0 ; i < MAX\_N ; i++) {

g[i].clear();

dist[i] = active[i] = excess[i] = 0;

}

}

void AddEdge(int from, int to, int cap) {

g[from].push\_back(Edge(from, to, cap, 0, g[to].size()));

if (from == to) g[from].back().index++;

g[to].push\_back(Edge(to, from, 0, 0, g[from].size() - 1));

}

void Enqueue(int v) {

if (!active[v] && excess[v] > 0) { active[v] = true; q.push(v); }

}

void Push(Edge &e) {

int amt = int(min(excess[e.from], ll(e.cap - e.flow)));

if (dist[e.from] <= dist[e.to] || amt == 0) return;

e.flow += amt;

g[e.to][e.index].flow -= amt;

excess[e.to] += amt;

excess[e.from] -= amt;

Enqueue(e.to);

}

void Gap(int k) {

for (int v = 0; v < n; v++) {

if (dist[v] < k) continue;

CNT[dist[v]]--;

dist[v] = max(dist[v], n+1);

CNT[dist[v]]++;

Enqueue(v);

}

}

void Relabel(int v) {

CNT[dist[v]]--;

dist[v] = 2\*n;

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

if (g[v][i].cap - g[v][i].flow > 0)

dist[v] = min(dist[v], dist[g[v][i].to] + 1);

CNT[dist[v]]++;

Enqueue(v);

}

void Discharge(int v) {

for (int i = 0; excess[v] > 0 && i < g[v].size(); i++) Push(g[v][i]);

if (excess[v] > 0) {

if (CNT[dist[v]] == 1)

Gap(dist[v]);

else

Relabel(v);

}

}

void push\_relabel() {

CNT[0] = n-1;

CNT[n] = 1;

dist[s] = n;

active[s] = active[t] = true;

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

excess[s] += g[s][i].cap;

Push(g[s][i]);

}

while (!q.empty()) {

int v = q.front();

q.pop();

active[v] = false;

Discharge(v);

}

ll totflow = 0;

for (int i = 0; i < g[s].size(); i++) totflow += g[s][i].flow;

maxFlow = totflow;

}

#### Min Cost Max Flow

//O(Unknown) 0-base

struct Edge {

int to, f, cap, cost, rev;

};

int n,m,s,t;

int prio[MAX\_N], curflow[MAX\_N], prevedge[MAX\_N], prevnode[MAX\_N], q[MAX\_N], pot[MAX\_N];

bool inqueue[MAX\_N];

vector<Edge> graph[MAX\_N];

void CLEAR() {

for(int i = 0 ; i < MAX\_N ; i++) {

prio[i] = curflow[i] = prevedge[i] = prevnode[i] = q[i] = pot[i] = inqueue[i] = 0;

graph[i].clear();

}

}

void addEdge(int s, int t, int cap, int cost) {

Edge a = {t, 0, cap, cost, graph[t].size()};

Edge b = {s, 0, 0, -cost, graph[s].size()};

graph[s].push\_back(a);

graph[t].push\_back(b);

}

void bellmanFord(int s, int dist[]) {

for(int i = 0 ; i< MAX\_N ; i++)

dist[i] = INF;

dist[s] = 0;

int qt = 0;

q[qt++] = s;

for (int qh = 0; (qh - qt) % n != 0; qh++) {

int u = q[qh % n];

inqueue[u] = false;

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

Edge &e = graph[u][i];

if (e.cap <= e.f) continue;

int v = e.to;

int ndist = dist[u] + e.cost;

if (dist[v] > ndist) {

dist[v] = ndist;

if (!inqueue[v]) {

inqueue[v] = true;

q[qt++ % n] = v;

}

}

}

}

}

pair<int, int> minCostFlow(int s, int t, int maxf) {

// bellmanFord can be safely commented if edges costs are non-negative

//bellmanFord(s, pot);

int flow = 0;

int flowCost = 0;

while (flow < maxf) {

priority\_queue<ll, vector<ll>, greater<ll> > q;

q.push(s);

for(int i = 0 ; i < MAX\_N ; i++)

prio[i] = INF;

prio[s] = 0;

curflow[s] = INF;

while (!q.empty()) {

ll cur = q.top();

int d = cur >> 32;

int u = cur;

q.pop();

if (d != prio[u])

continue;

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

Edge &e = graph[u][i];

int v = e.to;

if (e.cap <= e.f) continue;

int nprio = prio[u] + e.cost + pot[u] - pot[v];

if (prio[v] > nprio) {

prio[v] = nprio;

q.push(((ll) nprio << 32) + v);

prevnode[v] = u;

prevedge[v] = i;

curflow[v] = min(curflow[u], e.cap - e.f);

}

}

}

if (prio[t] == INF)

break;

for (int i = 0; i < n; i++)

pot[i] += prio[i];

int df = min(curflow[t], maxf - flow);

flow += df;

for (int v = t; v != s; v = prevnode[v]) {

Edge &e = graph[prevnode[v]][prevedge[v]];

e.f += df;

graph[v][e.rev].f -= df;

flowCost += df \* e.cost;

}

}

return make\_pair(flow, flowCost);

}

#### Bipartite Matching

//O(VE) 1-base

int n1,n2,match[MAX\_N];

bool mark[MAX\_N];

vector<int>g[MAX\_N];

bool dfs(int v) {

if(mark[v]) return false;

mark[v] = true;

for(int u:g[v]) {

if(match[u] == -1 || dfs(match[u])) {

match[u] = v, match[v] = u;

return true;

}

}

return false;

}

void optimize() {

for(int i = 1 ; i <= n1 ; i++) {

for(int v:g[i]) {

if(match[v] == -1) {

match[i] = v, match[v] = i;

break;

}

}

}

}

void MATCH() {

memset(match, -1, sizeof match);

optimize();

for(int i = 1 ; i <= n1 ; i++) {

if(match[i] != -1) continue;

memset(mark, false, sizeof mark);

dfs(i);

}

#### Weighted Bipartite Matching

//O(Unknown) 0-base

//if we want max weight perfect matching then the edges that are not in the //graph should have -INF weight if we want just max weight matching then //the edges that are not in the graph should have 0 weight if we want min //wieght matching then we negative the edges weight

int a[MAX\_N][MAX\_N], ulable[MAX\_N], dlable[MAX\_N], n,m;

int umatch[MAX\_N], dmatch[MAX\_N], umark[MAX\_N], dmark[MAX\_N];

bool dfs(int k){

umark[k]=1;

for(int i=0; i<n; i++) if(dmark[i]==0 && ulable[k]+dlable[i]==a[k][i]){

dmark[i]=1;

bool done=0;

if(dmatch[i]==-1){

done=1;

}else{

if(dfs(dmatch[i])) done=1;

}

if(done){

umatch[k]=i;

dmatch[i]=k;

return 1;

}

}

return 0;

}

void mwmatching(){

for(int i = 0 ; i < MAX\_N ; i++)

ulable[i] = dlable[i] = 0;

for(int i=0; i<n; i++)

for(int j=0; j<n; j++)

ulable[i]=max(ulable[i], a[i][j]);

for(int i = 0 ; i < MAX\_N ; i++)

umatch[i] = dmatch[i] = -1;

for(int size=0; size<n; ){

bool done=1;

while(done){

done=0;

for(int i = 0 ; i < MAX\_N ; i++)

umark[i] = dmark[i] = 0;

for(int i=0; i<n; i++) if(umark[i]==0 && umatch[i]==-1)

if(dfs(i)){

done=1;

size++;

}

}

int eps=(int)(1e9);

for(int i=0; i<n; i++) if(umark[i])

for(int j=0; j<n; j++) if(!dmark[j])

eps=min(eps, ulable[i]+dlable[j]-a[i][j]);

for(int i=0; i<n; i++)

if(umark[i]) ulable[i]-=eps;

for(int i=0; i<n; i++)

if(dmark[i]) dlable[i]+=eps;

}

}

int main(){

for(int i=0; i<m; i++){

cin >>x >>y >>w;

a[x][y]=max(w, a[x][y]);

}

mwmatching();

int ans=0;

for(int i=0; i<n; i++)

ans+=a[i][umatch[i]];

cout <<ans <<endl;

#### Cut Vertex(Biconnected Component)

//O(E) no-base

ll n,m, par[MAX\_N],low[MAX\_N], height[MAX\_N] ,markV[MAX\_N],cnt;

vector<ll>v[MAX\_N];

vector<pair<ll,ll> >bcc[MAX\_N], tmp\_find;

bool mark[MAX\_N];

vector<ll>articulationPoints;

set<ll>what;

void CLEAR() {

tmp\_find.clear(); cnt = 1; bridge.clear();

for(int i = 0 ; i < MAX\_N ; i++)

bcc[i].clear(); v[i].clear() low[i] = par[i] = height[i] = mark[i] = markV[i] = 0

}

void FIND(pair<ll,ll>x) {

while(tmp\_find.size() > 0) {

pair<ll,ll>y = tmp\_find[tmp\_find.size()-1];

tmp\_find.pop\_back();

bcc[cnt].push\_back(y);

if(y == x || (y.first == x.second && y.second == x.first))

break;

}

cnt++;

}

void dfs(int u, int h) {

mark[u] = true;

low[u] = h;

height[u] = h;

int childCount = 0;

bool isArticulation = false;

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

int node = v[u][i];

if(!mark[node]) {

tmp\_find.push\_back(make\_pair(u,node));

par[node] = u;

dfs(node, h+1);

childCount++;

if(low[node] >= height[u]) {

FIND(make\_pair(u,node));

isArticulation = true;

}

low[u] = min(low[u], low[node]);

}

else if(node != par[u] && height[node] < height[u]) {

tmp\_find.push\_back(make\_pair(u,node));

low[u] = min(low[u], height[node]);

}

}

if((par[u] != 0 && isArticulation) || (par[u] == 0 && childCount > 1)) {

articulationPoints.push\_back(u);

markV[u] = true;

}

#### Cut Edge

//O(E) no-base

int n, m, par[MAX\_N], low[MAX\_N], height[MAX\_N] ;

bool mark[MAX\_N];

vector<int>g[MAX\_N];

vector<pair<int,int> >cutEdges;

//dfs(1,0)

void dfs(int v, int h) {

mark[v] = true;

low[v] = h;

height[v] = h;

for(int u:g[v]) {

if(!mark[u]) {

par[u] = v;

dfs(u, h+1);

if(low[u] > height[v])

cutEdges.push\_back({v, u});

low[v] = min(low[v], low[u]);

}

else if(u != par[v] && height[u] < height[v])

low[v] = min(low[v], height[u]);

}

}

#### LCA

//O(n log n) no-base

//dfs(0, 0)

const int MAXN=1e5+10,MAXL=20;

vector<int> g[MAXN];

int par[MAXN][MAXL],h[MAXN];

void dfs(int v,int p) {

par[v][0]=p;

for(int i=1;i<MAXL;i++)

par[v][i]=par[par[v][i-1]][i-1];

for(int u:g[v])

if(u!=p) {

h[u]=h[v]+1;

dfs(u,v);

}

}

int get\_par(int v,int h) {

for(int i=0;i<MAXL;i++)

if(h&(1<<i))

v=par[v][i];

return v;

}

int LCA(int v,int u) {

if(h[v]>h[u])

swap(v,u);

u=get\_par(u,h[u]-h[v]);

if(v==u)

return v;

for(int i=MAXL-1;i>=0;i--)

if(par[v][i]!=par[u][i]) {

v=par[v][i];

u=par[u][i];

}

return par[v][0];

}

#### Eulerian Tour

//O(E) no-base

//if the odd degree vertex is 2 or all degrees are even

//directional two vertex one out and one in

//edges are numbered from 1 to m

//g[i][j] = {k, l} means i-th vertex is connected with k-th vertex through l-th edge

int n, m;

bool mark[MAX\_N];

vector<pair<int, int> >g[MAX\_N];

vector<int>ans;

void Euler(int v) {

while(g[v].size() > 0) {

pair<int, int> u = g[v].back();

g[v].pop\_back();

if(!mark[u.second]) {

mark[u.second] = 1;

Euler(d);

}

}

ans.push\_back(u);

}

# Data Structures

#### Ordered Set

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

using namespace \_\_gnu\_pbds;

typedef tree<int,null\_type,less<int>,rb\_tree\_tag,tree\_order\_statistics\_node\_update> Tree;

Tree t;

int main() {

int n;

cin>>n;

for(int i=0;i<n;i++) {

int x;

cin>>x;

t.insert(x);

}

int x;

while(cin>>x) {

int index=t.order\_of\_key(x);

int value=\*t.find\_by\_order(x);

cout<<index<<endl;

}

#### Heavy Light

vector <int> g[MAXN];

int par[MAXN],h[MAXN],sz[MAXN],up[MAXN],st[MAXN],en[MAXN],a[MAXN];

int T=0,n, seg[MAXN\*4];

void dfs\_make(int v,int p=0) {

par[v]=p;

if(v!=0) h[v]=h[p]+1;

sz[v]=1;

int ind=0,Max=0,pind=-1;

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

int u=g[v][i];

if(u!=p) {

dfs\_make(u,v);

sz[v]+=sz[u];

if(sz[u]>Max)ind=i,Max=sz[u];

}

else pind=i;

}

if(pind!=-1) {

swap(g[v][pind],g[v][g[v].size()-1]);

g[v].pop\_back();

}

if(g[v].size()) swap(g[v][0],g[v][ind]);

}

void dfs\_hld(int v) {

st[v]=T++;

if(g[v].empty()==0) {

up[g[v][0]]=up[v];

dfs\_hld(g[v][0]);

for(int i=1;i<g[v].size();i++){

int u=g[v][i];

up[u]=u;

dfs\_hld(u);

}

}

en[v]=T;

}

inline bool cont(int v,int u) {

return (st[u]>=st[v] and st[u]<en[v]);

}

int lca(int v,int u) {

if(cont(v,u)) return v;

if(cont(u,v)) return u;

int ans1=par[up[v]];

while(!cont(ans1,u)) ans1=par[up[ans1]];

int ans2=par[up[u]];

while(!cont(ans2,u)) ans2=par[up[ans2]];

if(h[ans1]<h[ans2]) return ans2;

else return ans1;

}

void add(int s,int e,int ind,int i,int val){

if(s>i or e<=i) return;

if(e==s+1) {

seg[ind]+=val;

return;

}

int mid=(s+e)/2;

add(s,mid,left(ind),i,val), add(mid,e,right(ind),i,val);

seg[ind]=seg[left(ind)]+seg[right(ind)];

}

int fin(int s,int e,int ind,int x,int y) {

if(s>=y or e<=x) return 0;

if(x<=s and e<=y) return seg[ind];

int mid=(s+e)/2;

return fin(s,mid,left(ind),x,y) +fin(mid,e,right(ind),x,y);

}

int calc(int Par,int v) {

if(Par==v) return a[v];

int ret=0,last=-1;

while(st[v]>st[Par]) {

if(st[up[v]]>st[Par]) ret+=fin(0,n,1,st[up[v]],st[v]+1);

else break;

v=par[up[v]];

}

ret+=fin(0,n,1,st[Par],st[v]+1);

return ret;

}

int main() {

cin>>n;

for(int i=0;i<n;i++) cin>>a[i];

for(int i=1;i<n;i++) {

int v,u;

cin>>v>>u;

g[v].push\_back(u);

g[u].push\_back(v);

}

dfs\_make(0);

up[0]=0;

dfs\_hld(0);

for(int i=0;i<n;i++) add(0,n,1,st[i],a[i]);

int m; cin>>m;

for(int i=0;i<m;i++) {

int v,u;

cin>>v>>u;

int l=lca(v,u);

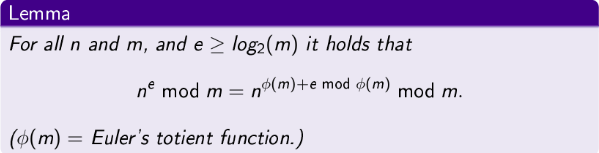
cout<<calc(l,v)+calc(l,u)-a[v]<<endl;

}

}

# Math

#### Power Lemma



#### Wilson

(p-1)! mod p = -1

#### AKS

#### Newton’s Method

Solve equations with one variable start from a random guess

By solve it means find x where f(x) = 0

for square root

#### Extended Euclid- Chinese Remainder

//O(log(n)) no-base

ll x,y,d;

//check for a == 0& b == 0 separately

//the answer is less for a and more for b

//Solve a\*x + b\*y = d where d = gcd(a,b)

//answers of X = x+(b/d)\*n , answers of Y = y-(a/d)\*n

void extendedEuclid(ll a, ll b) {

if (b == 0) x = 1; y = 0; d = a; return;

extendedEuclid(b, a % b);

ll x1 = y; ll y1 = x - (a / b) \* y; x = x1;y = y1;

}

int inv(int a, int m) {

int m0 = m, t, q; int x0 = 0, x1 = 1; if (m == 1) return 0;

while (a > 1)q = a / m; t = m; m = a % m, a = t; t = x0; x0 = x1 - q \*x0; x1 = t;

if (x1 < 0) x1 += m0; return x1;

}

//O(k) k is size of num[] and rem[]. Returns the smallest

// number x such that:

// x % num[0] = rem[0],

// x % num[1] = rem[1],...

// x % num[k-1] = rem[k-1]

// Assumption: Numbers in num[] are pairwise coprime

int chinese\_remainder(int num[], int rem[], int k) {

int prod = 1;

for (int i = 0; i < k; i++)

prod \*= num[i];

int result = 0;

for (int i = 0; i < k; i++) {

int pp = prod / num[i];

result += rem[i] \* inv(pp, num[i]) \* pp;

}

return result % prod;

}

// finds all solutions to ax = b (mod n)

vector<int> modular\_linear\_equation\_solver (int a, int b, int n){

int x, y;

vector<int> solutions;

int d = extended\_euclid (a, n, x, y);

if (b%d == 0){

x = mod (x\*(b/d), n);

for (int i = 0; i < d; i++)

solutions.push\_back (mod (x + i\*(n/d), n));

}

return solutions;

}

#### Miller

//O(iteration) no-base

ll modulo(ll x, ll y, ll Mod){

ll ret=1;

for(; y!=0; y/=2){

if(y%2) ret=(ret\*x)%Mod;

x=(x\*x)%Mod;

}

return ret;

}

bool Miller(ll p,int iteration){

if(p<2) return 0 if(p==2)return 1; if(p%2==0) return 0;

ll s=p-1; while(s%2==0)s/=2;

for(int i=0; i<iteration; i++){

ll a=rand()%(p-1)+1, temp=s;

ll mod=modulo(a, temp, p);

while(temp!=p-1 && mod!=1 && mod!=p-1){

mod=(mod\*mod)%p;

temp\*=2;

}

if(mod!=p-1 && temp%2==0)

return 0;

}

return 1;

}

#### Linear System

//O(n^3) 0-base

// Gauss-Jordan elimination with full pivoting.

// Uses:

// (1) solving systems of linear equations (AX=B)

// (2) inverting matrices (AX=I)

// (3) computing determinants of square matrices

// INPUT: a[][] = an nxn matrix

// b[][] = an nxm matrix solve m different equation

// OUTPUT: X = an nxm matrix (stored in b[][])

// A^{-1} = an nxn matrix (stored in a[][])

// returns determinant of a[][]

const double EPS = 1e-10;

typedef vector<int> VI;

typedef double T;

typedef vector<T> VT;

typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {

const int n = a.size();

const int m = b[0].size();

VI irow(n), icol(n), ipiv(n);

T det = 1;

for (int i = 0; i < n; i++) {

int pj = -1, pk = -1;

for (int j = 0; j < n; j++) if (!ipiv[j])

for (int k = 0; k < n; k++) if (!ipiv[k])

if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }

if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; return 0; }

ipiv[pk]++;

swap(a[pj], a[pk]);

swap(b[pj], b[pk]);

if (pj != pk) det \*= -1;

irow[i] = pj;

icol[i] = pk;

T c = 1.0 / a[pk][pk];

det \*= a[pk][pk];

a[pk][pk] = 1.0;

for (int p = 0; p < n; p++) a[pk][p] \*= c;

for (int p = 0; p < m; p++) b[pk][p] \*= c;

for (int p = 0; p < n; p++) if (p != pk) {

c = a[p][pk];

a[p][pk] = 0;

for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] \* c;

for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] \* c;

}

}

for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {

for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);

}

return det;

}

int main() {

double A[MAX\_N][MAX\_N],B[MAX\_N][MAX\_N];

n = 2, m = 1;

VVT a(n), b(n);

for (int i = 0; i < n; i++) {

a[i] = VT(A[i], A[i] + n);

b[i] = VT(B[i], B[i] + m);

}

double det = GaussJordan(a, b);

cout << "Determinant: " << det << endl;

cout << "Inverse: " << endl;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++)

cout << a[i][j] << ' ';

cout << endl;

}

cout << "Solution: " << endl;

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++)

cout << b[i][j] << ' ';

cout << endl;

}

}

#### Catalan Number

#### FFT

//O(n log n) 0-base

typedef complex<double> ftype;

const double pi = acos(-1);

const int maxn = 1 << 17;

ftype w[maxn];

void init() {

for(int i = 0; i < maxn; i++) {

w[i] = polar(1., 2 \* pi / maxn \* i);

}

}

template<typename T>

void fft(T \*in, ftype \*out, int n, int k = 1) {

if(n == 1) {

\*out = \*in;

return;

}

int t = maxn / n;

n >>= 1;

fft(in, out, n, 2 \* k);

fft(in + k, out + n, n, 2 \* k);

for(int i = 0, j = 0; i < n; i++, j += t) {

ftype t = w[j] \* out[i + n];

out[i + n] = out[i] - t;

out[i] += t;

}

}

vector<ftype> evaluate(vector<int> p) {

while(\_\_builtin\_popcount(p.size()) != 1) {

p.push\_back(0);

}

vector<ftype> res(p.size());

fft(p.data(), res.data(), p.size());

return res;

}

vector<int> interpolate(vector<ftype> p) {

int n = p.size();

vector<ftype> inv(n);

fft(p.data(), inv.data(), n);

vector<int> res(n);

for(int i = 0; i < n; i++) {

res[i] = round(real(inv[i]) / n);

}

reverse(begin(res) + 1, end(res));

return res;

}

void align(vector<int> &a, vector<int> &b) {

int n = a.size() + b.size() - 1;

while(a.size() < n) {

a.push\_back(0);

}

while(b.size() < n) {

b.push\_back(0);

}

}

vector<int> poly\_multiply(vector<int> a, vector<int> b) {

align(a, b);

auto A = evaluate(a);

auto B = evaluate(b);

for(int i = 0; i < A.size(); i++) {

A[i] \*= B[i];

}

return interpolate(A);

}

const int base = 10;

vector<int> normalize(vector<int> c) {

int carry = 0;

for(auto &it: c) {

it += carry;

carry = it / base;

it %= base;

}

while(carry) {

c.push\_back(carry % base);

carry /= base;

}

return c;

}

//multiple of two number

vector<int> multiply(vector<int> a, vector<int> b) {

return normalize(poly\_multiply(a, b));

}

int main() {

init(); //coef of x^0 x^1 ...

vector<int>a, b, ans;

ans = poly\_multiply(a, b);

while(ans.back() == 0) ans.pop\_back();

}

# Geometry

#### Simplex Algorithm

//O(unknown) 0-base

//This is a simplex solver. Given m x n matrix A, m-vector b, n-vector c,

//finds n-vector x such that

//A x <= b (component-wise)

//maximizing

//< x , c >

//where <x,y> is the dot product of x and y.

typedef long double DOUBLE;

typedef vector<DOUBLE> VD;

typedef vector<VD> VVD;

typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {

int m, n;

VI B, N;

VVD D;

LPSolver(const VVD &A, const VD &b, const VD &c) :

m(b.size()), n(c.size()), N(n+1), B(m), D(m+2, VD(n+2)) {

for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];

for (int i = 0; i < m; i++) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }

for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }

N[n] = -1; D[m+1][n] = 1;

}

void Pivot(int r, int s) {

for (int i = 0; i < m+2; i++) if (i != r)

for (int j = 0; j < n+2; j++) if (j != s)

D[i][j] -= D[r][j] \* D[i][s] / D[r][s];

for (int j = 0; j < n+2; j++) if (j != s) D[r][j] /= D[r][s];

for (int i = 0; i < m+2; i++) if (i != r) D[i][s] /= -D[r][s];

D[r][s] = 1.0 / D[r][s];

swap(B[r], N[s]);

}

bool Simplex(int phase) {

int x = phase == 1 ? m+1 : m;

while (true) {

int s = -1;

for (int j = 0; j <= n; j++) {

if (phase == 2 && N[j] == -1) continue;

if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] < N[s]) s = j;

}

if (D[x][s] >= -EPS) return true;

int r = -1;

for (int i = 0; i < m; i++) {

if (D[i][s] <= 0) continue;

if (r == -1 || D[i][n+1] / D[i][s] < D[r][n+1] / D[r][s] ||

D[i][n+1] / D[i][s] == D[r][n+1] / D[r][s] && B[i] < B[r]) r = i;

}

if (r == -1) return false;

Pivot(r, s);

}

}

DOUBLE Solve(VD &x) {

int r = 0;

for (int i = 1; i < m; i++) if (D[i][n+1] < D[r][n+1]) r = i;

if (D[r][n+1] <= -EPS) {

Pivot(r, n);

if (!Simplex(1) || D[m+1][n+1] < -EPS) return -numeric\_limits<DOUBLE>::infinity();

for (int i = 0; i < m; i++) if (B[i] == -1) {

int s = -1;

for (int j = 0; j <= n; j++)

if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j] < N[s]) s = j;

Pivot(i, s);

}

}

if (!Simplex(2)) return numeric\_limits<DOUBLE>::infinity();

x = VD(n);

for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n+1];

return D[m][n+1];

}

};

int main() {

const int m = 4;

const int n = 3;

DOUBLE \_A[m][n] = {

{ 6, -1, 0 },

{ -1, -5, 0 },

{ 1, 5, 1 },

{ -1, -5, -1 } };

DOUBLE \_b[m] = { 10, -4, 5, -3 };

DOUBLE \_c[n] = { 1, -1, 0 };

VVD A(m);

VD b(\_b, \_b + m);

VD c(\_c, \_c + n);

for (int i = 0; i < m; i++) A[i] = VD(\_A[i], \_A[i] + n);

LPSolver solver(A, b, c);

VD x;

DOUBLE value = solver.Solve(x);

cerr << "VALUE: "<< value << endl;

cerr << "SOLUTION:";

for (size\_t i = 0; i < x.size(); i++) cerr << " " << x[i];

cerr << endl;

return 0;

}

#### Closest Pair

//O(n logn) 1-base

struct point {

ll x,y;

};

int t,n;

point p[MAX\_N],p1[MAX\_N];

vector<point>v;

bool cmp(point a,point b) {

if(a.x < b.x) return true;

if(a.x > b.x) return false;

return a.y <= b.y;

}

bool cmp1(point a,point b) {

if(a.y < b.y) return true;

if(a.y > b.y) return false;

return a.x <= b.x;

}

ll dis(point a,point b) {

return (a.y-b.y)\*(a.y-b.y)+(a.x-b.x)\*(a.x-b.x);

}

pair<point,point>strip\_closest() {

ll MIN = INF;

point a,b;

for(int i = 0 ; i < v.size() ; i++) {

for(int j = i+1 ; j <= min(i+7,int(v.size())-1) ; j++) {

if(dis(v[i],v[j]) < MIN) {

MIN = dis(v[i],v[j]);

a = v[i];

b = v[j];

}

}

}

return make\_pair(a,b);

}

pair<point,point> closest\_pair(int l,int r) {

if(l == r) return make\_pair(p[1],p[2]);

if(l == r-1) return make\_pair(p[l],p[r]);

int mid = (l+r)/2;

pair<point,point>ret;

pair<point,point>a = closest\_pair(l,mid);

pair<point,point>b = closest\_pair(mid+1,r);

ll d;

if(dis(a.first,a.second) < dis(b.first,b.second)) {

ret = a;

d = dis(a.first,a.second);

}

else {

ret = b;

d = dis(b.first,b.second);

}

v.clear();

for(int i = l ; i <= r ; i++)

if(abs(p1[i].x -p1[mid].x) <= d) v.push\_back(p1[i]);

pair<point,point>c = strip\_closest();

if(dis(c.first,c.second) < dis(ret.first,ret.second)) ret = c;

return ret;

}

int main() {

scanf("%d",&n);

for(int i = 1 ; i <= n ; i++) {

scanf("%lld %lld",&p[i].x,&p[i].y);

p1[i] = p[i];

}

sort(p+1,p+n+1,cmp);

sort(p1+1,p1+n+1,cmp1);

pair<point,point> ans = closest\_pair(1,n);

}

#### Areas and Angles

double INF = 1e100;

double EPS = 1e-8;

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){ return PT(x-p.x,y-p.y); }

PT operator+ (const PT &p){ return PT(x+p.x,y+p.y); }

PT operator\* (double c){ return PT(x\*c,y\*c); }

PT operator/ (double c){ 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 dist (PT p, PT q) { return sqrt( dist2(p, q) ); }

double cross (PT p, PT q){ return p.x\*q.y-p.y\*q.x; }

// 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));

}

// rotate p1 around p0 clockwise, by angle a

PT RotateC(PT p0, PT p1, double a) {

p1 = p1-p0;

return p0 + PT(cos(a)\*p1.x-sin(a)\*p1.y,

sin(a)\*p1.x+cos(a)\*p1.y);

}

// p1->p2 line, reflect p3 to get r.

PT reflect(PT& p1, PT& p2, PT p3) {

if(dist(p1, p3)<EPS) {return p3;}

double a=dot(p2-p1,p3-p1)/(dist(p1,p2)\*dist(p1,p3));

a=acos(a);

return RotateC(p1, p3, -2.0\*a);

}

double SignedTriArea (PT a, PT b, PT c) {

return( (a.x\*b.y - a.y\*b.x + a.y\*c.x

- a.x\*c.y + b.x\*c.y - c.x\*b.y) / 2.0 );

}

double SignedArea (vector<PT> v){

double area = 0;

for (int i = 0; i < v.size(); i++){

int j = (i+1) % v.size();

area += v[i].x\*v[j].y - v[j].x\*v[i].y;

}

return area / 2.0;

}

#### Lines Points Intersect

// Given three colinear points p, q, r, the function checks if

// point q lies on line segment 'pr'

bool onSegment(PT p, PT q, PT r) {

if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&

q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))

return true;

return false;

}

//returns true if line segment 'p1q1' and 'p2q2' intersect.

// oriention code is in convex hull ( change Point to PT first! )

bool segmentIntersect(PT p1, PT q1, PT p2, PT q2) {

int o1 = orientation(p1, q1, p2);

int o2 = orientation(p1, q1, q2);

int o3 = orientation(p2, q2, p1);

int o4 = orientation(p2, q2, q1);

if (o1 != o2 && o3 != o4)

return true;

// p1, q1 and p2 are colinear and p2 lies on segment p1q1

if (o1 == 0 && onSegment(p1, p2, q1)) return true;

if (o2 == 0 && onSegment(p1, q2, q1)) return true;

if (o3 == 0 && onSegment(p2, p1, q2)) return true;

if (o4 == 0 && onSegment(p2, q1, q2)) return true;

return false;

}

// 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 the distance from AB to C

//if isSegment is true, AB is a segment, not a line.

double LinePointDist(PT A, PT B, PT C, bool isSegment){

double dd = cross(B-A,C-A) / dist(A,B);

if(isSegment){

int dot1 = dot(B-A,C-B);

if(dot1 > 0)return dist(B,C);

int dot2 = dot(A-B,C-A);

if(dot2 > 0)return dist(A,C);

}

return abs(dd);

}

// 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 two lines 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-c,d-c)) < EPS;

}

// compute intersection of line passing through a and b

// with line passing through c and d, assuming that unique

// intersection exists ( check collinear parallel )

PT ComputeLineIntersection (PT a, PT b, PT c, PT d){

b=b-a; d=c-d; c=c-a;

if (dot(b,b) < EPS) return a;

if (dot(d,d) < EPS) return c;

return a + b\*cross(c,d)/cross(b,d);

}

// the relation of the point p and the segment p1->p2.

// 1 if point is on the segment; 0 if not on the line;

// -1 if on the line but not on the segment

int pAndSeg(PT& p1, PT& p2, PT& p) {

double s=abs(SignedTriArea(p, p1, p2));

if(s>EPS) return(0);

double sg=(p.x-p1.x)\*(p.x-p2.x);

if(sg>EPS) return(-1);

sg=(p.y-p1.y)\*(p.y-p2.y);

if(sg>EPS) return(-1);

return(1);

}

#### Centers and Centroids

// 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));

}

// point generated by altitudes ( assuming it is triangle )

PT ComputeHcenter( PT p1, PT p2, PT p3 ) {

PT a1 = ProjectPointLine( p2, p3, p1 );

PT a2 = ProjectPointLine( p1, p3, p2 );

return ComputeLineIntersection( p1, a1, p2, a2 );

}

// point generated by circumscribed circle ( assuming tri )

PT ComputeCenter( PT p1, PT p2, PT p3 ) {

PT a1 = (p2+p3)\*0.5;

PT a2 = (p1+p3)\*0.5;

PT b1( a1.x - (p3.y-p2.y), a1.y + (p3.x-p2.x) );

PT b2( a2.x - (p3.y-p1.y), a2.y + (p3.x-p1.x) ) ;

return ComputeLineIntersection( a1, b1, a2, b2 );

}

PT ComputeCentroid (vector<PT> v){

double cx = 0, cy = 0;

double scale = 6.0 \* SignedArea (v);

for (int i = 0; i < v.size(); i++){

int j = (i+1) % v.size();

cx += (v[i].x+v[j].x)\*(v[i].x\*v[j].y-v[j].x\*v[i].y);

cy += (v[i].y+v[j].y)\*(v[i].x\*v[j].y-v[j].x\*v[i].y);

}

PT res; res.x = cx/scale; res.y = cy/scale;

return res;

}

// angle bisection ( assuming tri )

PT ComputebBcenter( PT p1, PT p2, PT p3 ) {

double s1, s2, s3;

s1 = dist( p2, p3 );

s2 = dist( p1, p3 );

s3 = dist( p1, p2 );

double rt = s2/(s2+s3);

PT a1 = p2\*rt+p3\*(1.0-rt);

rt = s1/(s1+s3);

PT a2 = p1\*rt+p3\*(1.0-rt);

return ComputeLineIntersection( a1,p1, a2,p2 );

}

#### Point in Polygon

// 1 if p is in pv; 0 outside; -1 on the polygon

int PointInPolygon(vector<PT> pv, PT p)

{

int n=pv.size(), j; pv.push\_back(pv[0]);

for(int i=0;i<n;i++)

if(pAndSeg(pv[i], pv[i+1], p)==1) return(-1);

for(int i=0;i<n;i++) pv[i] = pv[i]-p;

p.x=p.y=0.0; double a, y;

while(1) {

a=(double)rand()/10000.00;

j=0;

for(int i=0;i<n;i++) {

pv[i] = RotateCCW(pv[i], a);

if(abs(pv[i].x)<EPS) j=1;

}

if(j==0) {

pv[n]=pv[0];

j=0;

for(int i=0;i<n;i++)

if(pv[i].x\*pv[i+1].x < -EPS) {

y=pv[i+1].y-pv[i+1].x\*(pv[i].y-pv[i+1].y)/(pv[i].x-pv[i+1].x);

if(y>0) j++;

}

return(j%2);

}

}

return 1;

}

#### Big Intersections

// compute intersection of line through points a and b with

// circle centered at c with radius r > 0

vector<PT> CLIntersection (PT a, PT b, PT c, double r){

vector<PT> ret;

PT d = b-a;

double D = cross(a-c,b-c);

double e = r\*r\*dot(d,d)-D\*D;

if (e < 0) return ret;

e = sqrt(e);

ret.push\_back (c+PT(D\*d.y+(d.y>=0?1:-1)\*d.x\*e,-D\*d.x+fabs(d.y)\*e)/dot(d,d));

if (e > 0)

ret.push\_back (c+PT(D\*d.y-(d.y>=0?1:-1)\*d.x\*e,-D\*d.x-fabs(d.y)\*e)/dot(d,d));

return ret;

}

// compute intersection of circle centered at a with radius r

// with circle centered at b with radius R

vector<PT> CCIntersection (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;

}

vector<PT> PPIntersection(vector<PT>& p1, vector<PT>& p2) {

vector<PT> pts;

PT pp; pts.clear();

int m=p1.size(), n=p2.size();

for(int i=0;i<m;i++)

if(PointInPolygon(p2, p1[i])!=0) pts.push\_back(p1[i]);

for(int i=0;i<n;i++)

if(PointInPolygon(p1, p2[i])!=0) pts.push\_back(p2[i]);

if(m>1 && n>1)

for(int i=0;i<m;i++)

for(int j=0;j<n;j++)

if( !LinesParallel(p1[i], p1[(i+1)%m], p2[j], p2[(j+1)%n]) ) {

pp = ComputeLineIntersection(p1[i], p1[(i+1)%m], p2[j], p2[(j+1)%n]);

if(pAndSeg(p1[i], p1[(i+1)%m], pp)!=1) continue;

if(pAndSeg(p2[j], p2[(j+1)%n], pp)!=1) continue;

pts.push\_back(pp);

}

if(pts.size()<=1)

pts.clear();

return pts;

}

// cut the convex polygon pol along line p1->p2;

// pol1 are the resulting polygon on the left side, pol2 on the right.

void cutPoly(vector<PT>& pol, PT& p1, PT& p2, vector<PT>& pol1, vector<PT>& pol2) {

pol1.clear(); pol2.clear();

int i, sg, n=pol.size();

PT q1,q2,r;

for(i=0;i<n;i++) {

q1=pol[i]; q2=pol[(i+1)%n];

sg=orientation(p1, p2, q1);

if(sg==0 || sg==2) pol1.push\_back(q1);

if(sg==0 || sg==1) pol2.push\_back(q1);

if( !LinesParallel(p1, p2, q1, q2) ) {

r = ComputeLineIntersection(p1, p2, q1, q2);

if(pAndSeg(q1, q2, r)==1) {

pol1.push\_back(r);

pol2.push\_back(r);

}

}

}

if(pol1.size()<=2) pol1.clear();

if(pol2.size()<=2) pol2.clear();

}

#### Convex Hull

//O(n log n) 0-base

struct PT { int x; int y; };

vector<PT> p; //PTs of the Polygon to be processed

vector<PT> S; //Contains the convex hull

const double PI = 2.0\*acos(0.0);

const double EPS = 1e-9; //too small/big?????

int orientation(PT p1, PT p2, PT p3) {

int val = (p2.y - p1.y) \* (p3.x - p2.x) -

(p2.x - p1.x) \* (p3.y - p2.y);

if (abs(val) < EPS) return 0; // colinear

return (val > 0)? 1: 2; // clock(1) or counterclockwise(2)

}

//Returns the square of distance

int distSq(PT p1, PT p2) {

return (p1.x - p2.x)\*(p1.x - p2.x) +

(p1.y - p2.y)\*(p1.y - p2.y);

}

bool cmp (PT p1, PT p2) {

int o = orientation(p1, p[0], p2);

if (o==0) return (distSq(p[0], p1) <= distSq(p[0], p2));

return (o==1);

}

void convexHull (int n=p.size()) {

// Find the bottommost-leftmost PT

int ymn = p[0].y, mn = 0;

for (int i = 1; i < n; i++) {

int y = p[i].y;

if ((y < ymn) || (ymn == y && p[i].x < p[mn].x))

ymn = p[i].y, mn = i;

}

swap(p[0], p[mn]);

sort(p.begin()+1, p.end(), cmp);

int m=1; //Removing collinears and same PTs

for (int i=1; i<n; i++) {

while (i < n-1 && orientation(p[0], p[i], p[i+1]) == 0)

i++;

p[m] = p[i]; m++;

}

if (m<3) return; // Go for the convex hull

S.push\_back(p[0]); S.push\_back(p[1]); S.push\_back(p[2]);

for (int i = 3; i < m; i++) {

// Keep removing top while the turn is not ccw

while (orientation(S[S.size()-2], S[S.size()-1], p[i]) != 2)

S.pop\_back();

S.push\_back(p[i]);

}

}

// return 0 if not convex, 1 if strictly convex,

// 2 if convex but there are points unnecesary

// this function does not work if the polygon is self intersecting

// in that case, compute the convex hull of v, and see if both have the same area

int isConvex( vector<PT>& v ) {

int c0=0, c1=0, c2=0, n=v.size();

for ( int i=0; i<n; i++ ) {

int j=(i+1)%n; k=(i+2)%n;

int s=orientation(v[i], v[j], v[k]);

if (s==0) c0++;

if (s==2) c1++;

if (s==1) c2++;

}

if(c1 && c2) return 0;

if(c0) return 2;

return 1;

}

#### Misc

//(PI/3) \* (H \*H) \* (3\*R-H)

//volume of part of sphere H is height fromt the buttom

//R is the radius of sphere

//\*Lattice Polygons and Pick’s Theorem: A(P) = I(P) + B(P)/2 − 1

// Where A(P) is the area of Polygon P, I(P) is the number of

// lattice points inside P and B(P) num of points on boundary.

//\*Check whether a polygon is convex: all three consecutive

// points in the polygon must make left-turns

// if visited in counter clockwise order.

//\*Van Goh's algorithm not mentioned. Keep its idea in mind.

// (ear-cutting and trianulation)

//\*The idea of using binary search instead of complex formulas.

# Dynamic Programming

#### LIS

//O(n log n) 0-base

//number of decreasing bags is size of LIS

//CAUTION: JUST THE SIZE NOT THE ORDER FOR ORDER USE VECTOR AND PAR

multiset<int>s;

int a[MAX\_N], n;

void LIS() {

for(int i=0 ; i<n ;i++) {

int x=a[i];

//for increasing lower to upper

multiset<int>::iterator it=s.lower\_bound(x);

if(it==s.end())

s.insert(x);

else {

s.erase(it);

s.insert(x);

}

}

}

#### Matrix DP

\* =

Works for 2D DP as well if dp[i][j] only be filled with dp[i-1][k]

Second dimension = n, build an n\*n matrix how to fill j from different k

# String

#### Hash

//O(n) 0-base

//be carefull of mod ! log MOD is mutlplied

const ll MOD = 999998727899999LL;

const ll P = 37;

ll h[MAX\_N],po[MAX\_N];

ll Hash(string s) {

h[0]=(s[0]-'a'+1);

for(int i=1;i<s.length();i++)

h[i]=(h[i-1]\*P+(s[i]-'a'+1))%MOD;

return h[s.length()-1];

}

ll mul(ll a, ll b) {

if(b == 0) return 0;

ll x = mul(a, b/2);

if(b%2) return (x+x+a)%MOD;

else return (x+x)%MOD;

}

ll calc(int s,int e){

if(s==0) return h[e];

ll ans=h[e];

ans-=mul(h[s-1], po[e-s+1]);

if(ans < 0) ans += MOD;

return ans;

}

void init() {

po[0]=1;

for(int i=1;i<MAX\_N;i++)

po[i]=(po[i-1]\*P)%MOD;

}

#### KMP

//O(K) 0-base

//Searches for the string w in the string s (of length k). Returns the

//0-based index of the first match (k if no match is found). Algorithm

void buildTable(string& w, vector <int>& t){

t = vector <int>(w.length());

int i = 2, j = 0;

t[0] = -1; t[1] = 0;

while(i < w.length()) {

if(w[i-1] == w[j]) { t[i] = j+1; i++; j++; }

else if(j > 0) j = t[j];

else { t[i] = 0; i++; }

}

}

int KMP(string& s, string& w) {

int m = 0, i = 0;

vector <int> t;

buildTable(w, t);

while(m+i < s.length()) {

if(w[i] == s[m+i]) {

i++;

if(i == w.length()) return m;

} else {

m += i-t[i];

if(i > 0) i = t[i];

}

}

return s.length();

}

# Misc

#### Ternary Search

//O(logn) no-base

//first increase then decrese

ll low = 0, high = INF;

ld ans = 0;

while(low<=high) {

ll lm = low+(high-low)/3;

ll rm = high-(high-low)/3;

ld lmval = check(lm);

ld rmval = check(rm);

if(lmval < rmval) {

ans = max(ans, lmval);

low = lm+1;

}

else {

ans = max(ans,d rmval);

high = rm-1;

}

}

#### Grid

