In The Name of GOD

Gharche Sokhari

#include <iostream>

#include <set>

#include <map>

#include <iomanip>

#include <cstdio>

#include <algorithm>

#include <vector>

#include <quque>

#include <dequeue>

#include <sstream>

#include <string>

#include <cstring>

#include <cmath>

**Graph :**

2-SAT(SCC) .................... 2

Bellman-Ford .................... 3

Floyd-Warshall .................... 3

Cut Edge .................... 4

Cut Vertex(biconnected) .................... 4

Dijekstra .................... 5

Eulerian Tour .................... 5

Flow (Ford-Fulkerson) .................... 5

Flow (Dinic) .................... 6

Mincost Maxflow ....................7

LCA .................... 8

Matching .................... 9

Weighted Matching .................... 10

Kruskal .................... 11

**Linear Algebra:**

Determinant & Equation & Inverse .................... 11

Rank of Matrix .................... 12

**Combination:**

Catalan Number .................... 13

Change Base .................... 13

Choose .................... 13

**Number Theory:**

Extended Euclid-Invert-Chinese Remainder .................... 14

Miller .................... 15

Pollard .................... 15

Power With Mod .................... 15

Prime Factor & Phi .................... 15

Sieve of Erothesten .................... 16

Newtons Method & Wilson .................... 16

Primes .................... 16

**Data Structure:**

Segment Tree(1D) .................... 16

Segment Tree(2D) .................... 17

MO .................... 17

**Dynamic Programming:**

LCS .................... 18  
LIS .................... 18

Linear DP .................... 18

**String Matching:**

Hash .................... 19

KMP .................... 19

**Geometry** .................... 19

**MISC:**

Stringstream .................... 24

NIM .................... 24

Merge Sort + Inversion .................... 24

Grid .................... 25

BigDecimal BigInteger .................... 25

**2-SAT(SCC):**

// booleans are 1 to n

//~i = i+n

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

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

vector<int>topologicalSort;

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

void tSort(int v) {

mark[v] = true;

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

if(!mark[g[v][i]])

tSort(g[v][i]);

topologicalSort.push\_back(v);

}

void dfs(int v) {

mark[v] = true;

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

verticesScc[v] = cnt;

for(int i = 0 ; i < g1[v].size() ; i++)

if(!mark[g1[v][i]])

dfs(g1[v][i]);

}

void SCC() {

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

if(!mark[i])

tSort(i);

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

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

mark[i] = false;

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

if(!mark[topologicalSort[i]]) {

cnt++;

dfs(topologicalSort[i]);

}

}

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 twoSat() {

n \*= 2;

SCC();

n/=2;

if(!check()) {

satisfy = false;

return;

}

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

mark[i] = false;

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

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

int f = topologicalSort[i];

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

mark[f] = true;

ans[f] = true;

}

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

mark[f-n] = true;

}

}

**Bellman-Ford:**

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

//first Node, second Node, weight

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

bool negativeCycle;

void bellmanFord() {

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

dis[i] = INF;

dis[1] = 0;

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

for(int j = 0 ; j < edges.size() ; j++) {

ll a = edges[j].first.first;

ll b = edges[j].first.second;

ll wieght = edges[j].second;

if(dis[a] + weight < dis[b])

dis[b] = dis[a]+wieght;

}

}

for(int j = 0 ; j < edges.size() ; j++) {

ll a = edges[j].first.first;

ll b = edges[j].first.second;

ll wieght = edges[j].second;

if(dis[a] + weight < dis[b])

negativeCycle = true;

}

}

we can solve

m 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]

**Floyd-Wrshall:**

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

//g[i][j] = INF, jahat dar

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]))

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

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

}

**Cut Edge:**

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

bool mark[MAX\_N];

vector<int>v[MAX\_N];

vector<pair<int,int> >articulationBridges;

//dfs(1,0)

void dfs(int u, int h) {

mark[u] = true;

low[u] = h;

height[u] = h;

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

int node = v[u][i];

if(!mark[node]) {

par[node] = u;

dfs(node, h+1);

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

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

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

}

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

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

}

}

**Cut Vertex(biconnected):**

int n, m, par[MAX\_N], low[MAX\_N], height[MAX\_N],cnt = 1;

bool mark[MAX\_N],markV[MAX\_N];

vector<int>v[MAX\_N];

vector<int>articulationPoints;

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

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++;

}

//dfs(1,0)

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;

}

}

**Dijekstra:**

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

//weight, node

vector<pair<ll,ll> >v[MAX\_N];

//weight, node

set<pair<ll,ll> >s;

void dijekstra(int start) {

for(int i = 1 ; i <= n ; i++) dis[i] = INF;

dis[start] = 0;

par[start] = -1;

s.insert(make\_pair(0,start));

while(s.size() > 0) {

pair<ll,ll>a = \*s.begin();

s.erase(\*s.begin());

ll distance = a.first;

ll node = a.second;

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

ll edgeWeight = v[node][i].first;

ll neigh = v[node][i].second;

if(distance + edgeWeight < dis[neigh]) {

s.erase(make\_pair(dis[neigh], neigh));

dis[neigh] = distance + edgeWeight;

par[neigh] = node;

s.insert(make\_pair(dis[neigh], neigh));

}

}

}

}

**Eulerian Tour:**

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

a b m[make\_pair(a,b)]++ , m[make\_pair(b,a)]++;

int n,mm;

vector<int>v[MAX\_N];

vector<int>ans;

map<pair<int,int>,int> m;

void Euler(int u) {

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

int d = v[u][i];

if(m[make\_pair(u,d)] > 0) {

m[make\_pair(u,d)]--;

m[make\_pair(d,u)]--;

Euler(d);

}

}

ans.push\_back(u);

}

**Flow (Ford-Fulkerson) :**

/\* Ford-Fulkerson O(maxFlow \* E)

bi-directional or not changes the c[i][j] & c[j][i]

c[i][j] = c[j][i] = capacity of that pipe

cf [i][j] = c[i][j] - f[i][j]

f[i][j] was unnecessary\*/

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

bool mark[MAX\_N];

bool bfs() {

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

mark[i] = false;

queue<int>q;

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 fordFulkerson() {

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;

}

}

**Flow (Dinic):**

//Dinic O(V^2\*E)

struct Edge {

int to,reverseIndex,cap,flow;

};

int n,m,s,t,maxFlow,dis[MAX\_N];

vector<Edge>g[MAX\_N];

bool bfs() {

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

dis[i] = INF;

queue<int>q;

q.push(s);

dis[s] = 0;

while(q.size() > 0) {

int v = q.front();

q.pop();

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

Edge x = g[v][i];

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;

}

}

**Mincost Maxflow:**

typedef long long ll;

typedef pair<int, int> pii;

const int MAX\_N = 100+10;

const int INF = 1e9;

struct Edge {

int to, f, cap, cost, rev;

};

//0 base

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;

}

}

}

}

}

pii 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);

}

int main() {

int time;

cin>>time;

for(int tt = 1 ; tt <= time ; tt++) {

CLEAR();

cin>>n>>m>>s>>t;

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

int a,b,c,d;

cin>>a>>b>>c>>d;

//first node , second node , capacity , cost

addEdge(a, b, c, d);

}

cout<<"Test Case "<<tt<<":"<<endl;

pii res = minCostFlow(s, t, INF);

int flow = res.first;

int flowCost = res.second;

cout<<flow<<" "<<flowCost<<endl<<endl;

}

}

**LCA:**

const int MAX\_N = 1e5+10;

const int MAX\_LOG = 20;

int n,q,par[MAX\_N][MAX\_LOG],h[MAX\_N];

vector<int>v[MAX\_N];

bool mark[MAX\_N];

// all parents are -1

void dfs(int u,int parent){

mark[u] = true;

par[u][0] = parent;

if(parent != -1)

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

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

if(par[u][i-1] != -1)

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

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

if(!mark[v[u][i]]) {

dfs(v[u][i],u);

}

}

}

int LCA(int x,int y) {

if(h[x] < h[y])

swap(x,y);

for(int i = MAX\_LOG -1 ; i >= 0 ; i--)

if(par[x][i] != -1 && h[par[x][i]] >= h[y])

x = par[x][i];

if(x == y)

return x;

for(int i = MAX\_LOG-1 ; i >= 0 ; i--)

if(par[x][i] != par[y][i]) {

x = par[x][i];

y = par[y][i];

}

return par[x][0];

}

**Matching:**

//1 to n1 first independent part

//n1+1 to n1+n2 second independent part

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

bool mark[MAX\_N];

vector<int>v[MAX\_N];

bool dfs(int u) {

if(mark[u])

return false;

mark[u] = true;

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

int w = v[u][i];

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

match[w] = u;

match[u] = w;

return true;

}

}

return false;

}

void optimize() {

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

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

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

match[i] = v[i][j];

match[v[i][j]] = i;

break;

}

}

}

}

void MATCH() {

for(int i = 1 ; i <= n1+n2 ; i++) match[i] = -1;

optimize();

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

if(match[i] != -1)

continue;

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

mark[j] = false;

dfs(i);

}

}

**Weighted Matching:**

/\*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];

int ulable[MAX\_N], dlable[MAX\_N];

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

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

int n, m;

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(){

cin >>n >>m;

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

int x, y, w;

cin >>x >>y >>w;

x--; y--;

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;

return 0;

}

**Kruskal:**

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

//Weight , firstNode , seconddNode

vector<pair<int,pair<int,int> > > edges;

vector<pair<int,pair<int,int> > > ans;

vector<int>cc[MAX\_N];

void join(int x,int y) {

x = par[x];

y = par[y];

if(cc[y].size() > cc[x].size()) swap(x,y);

for(int i = 0 ; i < cc[y].size() ; i++) {

int node = cc[y][i];

par[node] = x;

cc[x].push\_back(node);

}

cc[y].clear();

}

void kruskal() {

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

par[i] = i;

cc[i].push\_back(i);

}

sort(edges.begin(), edges.end());

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

int firstNode = edges[i].second.first;

int secondNode = edges[i].second.second;

if(par[firstNode] != par[secondNode]) {

join(firstNode, secondNode);

ans.push\_back(edges[i]);

}

}

}

**Determinant & Equation & Inverse:**

// 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

//

// Running time: O(n^3)

//

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

// b[][] = an nxm matrix

//

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

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

// returns determinant of a[][]

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

const double EPS = 1e-10;

const int MAX\_N = 100;

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() {

int n,m;

cin>>n>>m;

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

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

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

cin>>A[i][j];

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

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

cin>>B[i][j];

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;

}

}

**Rank of Matrix:**

int mat[MAX\_N][MAX\_N],R,C;

void swap(int row1, int row2,int col) {

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

int temp = mat[row1][i];

mat[row1][i] = mat[row2][i];

mat[row2][i] = temp;

}

}

int rankOfMatrix() {

int rank = C;

for (int row = 0; row < rank; row++) {

if (mat[row][row]) {

for (int col = 0; col < R; col++) {

if (col != row) {

double mult = (double)mat[col][row] /

mat[row][row];

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

mat[col][i] -= mult \* mat[row][i];

}

}

}

else {

bool reduce = true;

for (int i = row + 1; i < R; i++) {

if (mat[i][row]) {

swap(row, i, rank);

reduce = false;

break ;

}

}

if (reduce) {

rank--;

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

mat[i][row] = mat[i][rank];

}

row--;

}

}

return rank;

}

**Catalan Number:**

C(n) = C(0)\*C(n-1) + C(1)\*C(n-2) + ... + C(k)\*C(n-k-1) + ... + C(n-1)\*C(0)

C(n) = (1/(n+1)) choose(2n,n)

number or triangulation of convex

number of rooted binary trees on n+1 leaves

counting the number of right parentheses

2 = (1,1) & (2)

f(n,k) = part n with the biggest integer k

f(n,k) = f(n,k-1) + f(n-k,k)

**Change Base:**

ll changeBaseFromBaseToTen(vector<int> num,int base) {

ll co = 1;

ll ret = 0;

for(int i = num.size() -1 ; i >= 0 ; i--) {

ret += co\*num[i];

co \*= base;

}

return ret;

}

vector<int> changeBaseFromTenToBase(ll num,int base) {

vector<int>ret;

while(num != 0) {

ret.push\_back(num%base);

num /= base;

}

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

return ret;

}

**Choose:**

ll C[MAX\_N][MAX\_N];

void FILL\_CHOOSE() {

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

C[i][0] = 1;

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

C[i][i] = 1;

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

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

C[i][j] = (C[i-1][j-1] + C[i-1][j])%MOD;

}

int nCrModpLucas(int n,int r,int p){

if(r==0)

return 1;

int ni=n%p;

int ri=r%p;

return (nCrModpLucas(n/p,r/p,p)\*nCrModpLucas(ni,ri,p))%p;

}

**Extended Euclid-Invert-Chinese Remainder:**

ll x,y,d;

check for a == 0 && b == 0

the answer is less for a and more for b

d = gcd(a,b)

a\*x + b\*y = d extendedEuclid counts this x and y

a\*x + b\*y = c

c % d should be 0

two side of equation \* c/d

answers of X = x+(b/d)\*n

answers of Y = y-(a/d)\*n

n is a integer

if(n >= a)

then we should count n >= ceil(a);

for floor dont make int use floor

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;

}

// 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-2] = 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;

}

**Miller:**

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;

}

**Pollard:**

//one of the divisors of n

ll PollardRho(ll n) {

srand (time(NULL));

if (n==1) return n;

if (n % 2 == 0) return 2;

ll x = (rand()%(n-2))+2;

ll y = x;

ll c = (rand()%(n-1))+1;

ll d = 1;

while (d==1) {

x = (modular\_pow(x, 2, n) + c + n)%n;

y = (modular\_pow(y, 2, n) + c + n)%n;

y = (modular\_pow(y, 2, n) + c + n)%n;

d = \_\_gcd(abs(x-y), n);

if (d==n) return PollardRho(n);

}

return d;

}

**Power With Mod:**

ll POW(ll a, ll x) {

if(x == 1) return a;

if(x%2 == 0)

return POW((a\*a)%MOD, x/2);

else

return (a\*(POW((a\*a)%MOD, x/2)))%MOD;

}

**Prime Factor & Phi:**

//remember to call sieve before

vector<pair<ll,ll> > primeFactor(ll num) {

vector<pair<ll,ll> >ret;

for(int i = 0 ; i < p.size() && p[i]<= sqrt(num)+1 ; i++) {

ll t = 0;

while(num % p[i] == 0) {

t++;

num /= p[i];

}

if(t != 0)

ret.push\_back(make\_pair(p[i],t));

}

if(num != 1)

ret.push\_back(make\_pair(num,1));

return ret;

}

//attention to num = 1 in some case it should return 1

ll phi(ll num) {

if(num == 1)

return 0;

vector<pair<ll,ll> >tmp = primeFactor(num);

ll ret = num;

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

ll f = tmp[i].first;

ret \*= f-1;

ret /= f;

}

return ret;

}

**Sieve of Erothesten:**

//for memory optimization we can find radical(n) primes

//and then use the mark[radical(MAX\_N)] for the next primes and so on...

bool mark[MAX\_N];

vector<ll>p;

void sieve() {

mark[1] = true;

for(ll i = 2 ; i < MAX\_N ; i++) {

if(!mark[i]) {

p.push\_back(i);

for(ll j = i\*i ; j < MAX\_N ; j += i)

mark[j] = true;

}

}

}

**Newtons Method & Wilson:**

one variable equation

make a random guess like 1

x(1) = 1;

x(i+1) = x(i) - (F(x(i)) / F'(x(i)))

for square root

x(i+1) = 1/2(x(i) + n/x(i);

(p-1)! mod p = -1

**Primes:**

The largest prime smaller than 10^2 is 97.

The largest prime smaller than 10^3 is 997.

The largest prime smaller than 10^4 is 9973.

The largest prime smaller than 10^5 is 99991.

The largest prime smaller than 10^6 is 999983

The largest prime smaller than 10^7 is 9999991.

The largest prime smaller than 10^8 is 99999989.

The largest prime smaller than 10^9 is 999999937.

The largest prime smaller than 10^10 is 9999999967.

The largest prime smaller than 10^11 is 99999999977.

The largest prime smaller than 10^12 is 999999999989.

The largest prime smaller than 10^13 is 9999999999971.

The largest prime smaller than 10^14 is 99999999999973.

The largest prime smaller than 10^15 is 999999999999989.

The largest prime smaller than 10^16 is 9999999999999937.

The largest prime smaller than 10^17 is 99999999999999997.

The largest prime smaller than 10^18 is 999999999999999989.

**Segment Tree(1D) :**

//SEGMENT 1D for sum and add x to interval

const int MAX\_N = 2\*(1e5+10);

int n,seg[4\*MAX\_N],a[MAX\_N],flag[4\*MAX\_N];

int sum(int node\_num,int l,int r,int low,int high) {

if(high < l || low > r)

return 0;

low = max(low,l);

high = min(high,r);

if(l == low && r == high)

return seg[node\_num];

int mid = (l+r)/2;

return (flag[node\_num] \* (high-low+1))+(sum(2\*node\_num,l,mid,low,high)+sum(2\*node\_num+1,mid+1,r,low,high));

}

//add x to all numbers in [low,high]

int change(int node\_num,int l,int r,int low,int high,int x) {

if(high < l || low > r)

return seg[node\_num];

low = max(low,l);

high = min(high,r);

if(l == low && r == high) {

flag[node\_num] += x;

return seg[node\_num] = seg[node\_num] + ((r-l)+1)\*x;

}

int mid = (l+r)/2;

return seg[node\_num] = change(2\*node\_num,l,mid,low,high,x)+change(2\*node\_num+1,mid+1,r,low,high,x);

}

**Segment Tree(2D):**

//SEGMENT 2D FOR MAX\_MIN A RECTANGLE AND CHANGE A NUMBER

const int MAX\_N = 500+10;

const int INF = 1e9;

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

pair<int,int> seg[16\*MAX\_N\*MAX\_N];

//X az from tx to dx

//Y az from ly to ry

//we want to change (x,y) to new\_num

pair<int,int> change(int node\_num,int tx,int dx,int ly,int ry,int x,int y,int new\_num) {

if(tx > dx || ly > ry)

return make\_pair(0,INF);

if(x < tx || x > dx || y < ly || y > ry)

return seg[node\_num];

if(tx == dx && tx == x && ly == ry && ly == y)

return seg[node\_num] = make\_pair(new\_num,new\_num);

int midx = (tx+dx)/2;

int midy = (ly+ry)/2;

pair<int,int>A = change(4\*node\_num-2,tx,midx,ly,midy,x,y,new\_num);

pair<int,int>B = change(4\*node\_num-1,tx,midx,midy+1,ry,x,y,new\_num);

pair<int,int>C = change(4\*node\_num,midx+1,dx,ly,midy,x,y,new\_num);

pair<int,int>D = change(4\*node\_num+1,midx+1,dx,midy+1,ry,x,y,new\_num);

return seg[node\_num] = make\_pair(max(max(A.first,B.first),max(C.first,D.first)),min(min(A.second,B.second),min(C.second,D.second)));

}

//X az from tx to dx

//Y az from ly to ry

//we looking for X from a to b

//and Y from c to d

pair<int,int> max\_min(int node\_num,int tx,int dx,int ly,int ry,int a,int b,int c,int d) {

if(a > dx || b < tx || c > ry || d < ly || tx > dx || ly > ry)

return make\_pair(0,INF);

a = max(a,tx);

b = min(b,dx);

c = max(c,ly);

d = min(d,ry);

if(tx == a && dx == b && ly == c && ry == d)

return seg[node\_num];

int midx = (tx+dx)/2;

int midy = (ly+ry)/2;

pair<int,int>A = max\_min(4\*node\_num-2,tx,midx,ly,midy,a,b,c,d);

pair<int,int>B = max\_min(4\*node\_num-1,tx,midx,midy+1,ry,a,b,c,d);

pair<int,int>C = max\_min(4\*node\_num,midx+1,dx,ly,midy,a,b,c,d);

pair<int,int>D = max\_min(4\*node\_num+1,midx+1,dx,midy+1,ry,a,b,c,d);

return make\_pair(max(max(A.first,B.first),max(C.first,D.first)),min(min(A.second,B.second),min(C.second,D.second)));

}

**MO:**

1-

agar queri ha yek no bashan

va khasiate add va delete dashte bashim tuie arraye

L,R haro be in surat sort miknim ke agar L yeki tuie dasteie

radical N taE zudtar umade bud zudtar miumad

agar L/radical(n) barabar bud ba right sort miknim

az avalin queri shoro miknim for miznim

har dafe az curL ta cur R ro be L,R tabdil miknim ba

add o delete (yeki yeki harekat miknim)

2-

agar queri ha 2 no' bashan (update va sum)

ma update ha ro radical (Q) ta joda miknim

yek vector az queri haie update darim

har moghe be radical(Q) resid unaro tuie arraye piade miknim

mishe radical(Q)\*N(array size)

baraie sum ham jame arrayaro darim ta inja b joz queri haie

tuie vector k unaro emal miknim tu javab k mishe baz ham

raidcal(Q)\*Q

**LCS:**

string s,t;

int dp[MAX\_N][MAX\_N];

void LCS() {

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

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

dp[i][j] = 0;

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

for(int j = 1 ; j <= t.size() ; j++) {

if(s[i-1] == t[j-1])

dp[i][j] = dp[i-1][j-1] +1;

else

dp[i][j] = max(dp[i-1][j],dp[i][j-1]);

}

}

}  
**LIS:**

int n,par[MAX\_N],a[MAX\_N];

//par[i] = j means that in the LIS .... a[j] a[i] ...

//number can be large , can be equal

//strictly increasing

//for making just increasing remove cmp

bool cmp(pair<int,int> x, pair<int,int> y) {

if(x.first < y.first)

return true;

if(x.second > y.second)

return false;

return (x.second >= y.second);

}

vector<int> LIS() {

vector<pair<int,int> >v;

v.push\_back(make\_pair(a[1],1));

par[1] = -1;

for(int i = 2 ; i <= n ; i++) {

if(a[i] > v[v.size()-1].first) {

v.push\_back(make\_pair(a[i],i));

par[i] = v[v.size()-2].second;

continue;

}

int low = lower\_bound(v.begin(),v.end(),make\_pair(a[i],i),cmp)-v.begin();

if(v[low].first == a[i])

continue;

v[low] = make\_pair(a[i],i);

if(low == 0)

par[i] = -1;

else

par[i] = v[low-1].second;

}

vector<int>ret;

int cur = v[v.size()-1].second;

while(cur != -1) {

ret.push\_back(a[cur]);

cur = par[cur];

}

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

return ret;

}

**Linear DP:**

f(n) = a f(n-1) + b f(n-2) + c f(n-3)

f(n) a b c f(n-1)

f(n-1) = 1 0 0 \* f(n-2)

f(n-2) 0 1 0 f(n-3)

-->

f(n) a b c^n f(1)

f(n-1) = 1 0 0 \* f(2)

f(n-2) 0 1 0 f(3)

**Hash:**

const long long MOD = 1000000000+7;

const long long prime = 31;

const int MAX\_N = 100000+10;

long long p[MAX\_N];

void setUp(){

p[0]=1ll;

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

p[i]=(p[i-1]\*prime)%MOD;

}

}

string s;

long long hashTable[MAX\_N];

void hash(){

setUp();

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

hashTable[i]=(( ( (long long)(s[i]) \*p[i] ) % MOD ) + hashTable[i-1]) % MOD;

}

}

**KMP:**

const int MAX\_N = 100000+10;

int table[MAX\_N];

string s,t;

int n,m;

void setTable(){

// table[i] : max tool pishvandi az 0 ta i-1 tu reshte t be tori ke pasvande 0 ta i-1 ham hast

table[0]=-1;

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

table[i]=table[i-1];

while(table[i] != -1 && t[ table[i] ]!=t[i-1]){

table[i] = table[ table[i] ];

}

table[i]++;

}

}

void KMP(){

setTable();

int j=0;

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

while(j != -1 && s[i]!=t[j]){

j = table[j];

}

j++;

if(j == m){

cout<<i-m+1<<endl;

j = table[j];

}

}

}

**Geometry:**

struct CMP{

bool operator() (const pii& x, const pii& y) const{

if(x.first==y.first){

return (x.second>y.second);

}

return (x.first>y.first);

}

};

struct Point{

double x,y;

Point(double xx=0.0,double yy=0.0){

x=xx;

y=yy;

}

bool operator <(const Point &p) const {

return x<p.x || (x==p.x && y<p.y);

}

};

bool isZero(double x){

return ((x<1e-6) && (x>-1e-6));

}

// cosinos

double dot(Point a,Point b,Point c,Point d){

b.x-=a.x;

b.y-=a.y;

d.x-=c.x;

d.y-=c.y;

double ret = (b.x\*d.x)+(b.y\*d.y);

return ret;

}

// sinos

double cross(Point a,Point b,Point c,Point d){

b.x-=a.x;

b.y-=a.y;

d.x-=c.x;

d.y-=c.y;

double ret = (b.x\*d.y)-(b.y\*d.x);

return ret;

}

pair <double,double> lineLineInt;

bool lineLineIntersection(double a1,double b1,double c1,double a2,double b2,double c2){

double det = a1\*b2-a2\*b1;

if(isZero(det)){

return 0;

}else{

double x = (b2\*c1 - b1\*c2)/det;

double y = (a1\*c2 - a2\*c1)/det;

lineLineInt.first=x;

lineLineInt.second=y;

return 1;

}

}

double dis(Point a,Point b){

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

}

double linePointDis(Point a,Point b,Point c){

return abs((cross(a,b,a,c)/dis(a,b)));

}

double area(vector <Point> p){

double ret=cross(p[0],p[1],p[0],p[2]);

for(int i=3;i<p.size();i++){

ret = ret + cross(p[0],p[i-1],p[0],p[i]);

}

return abs(ret);

}

/////////////////////////////////////////////////////////////////////////

bool cmp(Point x,Point y){

if(isZero(x.x-y.x)){

return (x.y < y.y);

}else{

return (x.x < y.x);

}

}

vector <Point> convexHull(vector <Point> p){

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

vector <Point> u,l;

u.clear();

l.clear();

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

while(l.size()>=2){

if(cross(l[l.size()-1] ,l[l.size()-2] , l[l.size()-1] , p[i]) > 1e-6){

l.pop\_back();

}else{

break;

}

}

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

}

l.pop\_back();

for(int i=p.size()-1;i>=0;i--){

while(u.size()>=2){

if(cross(u[u.size()-1] ,u[u.size()-2] , u[u.size()-1] , p[i]) > 1e-6){

u.pop\_back();

}else{

break;

}

}

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

}

u.pop\_back();

vector <Point> ret;

ret.clear();

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

ret.push\_back(l[i]);

}

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

ret.push\_back(u[i]);

}

return ret;

}

////////////////////////////////////////////////////////////////////////

//Point in polygon

//complex

bool is\_strictly\_in(vector<point>& V, point p){

double total=0.0;

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

total+=arg((V[i]-p)/(V[(i+1)%(int)V.size()]-p));

}

if(total<0) total\*=-1.0;

return total>=3.0;

}

/////////////////////////////////////////////////////////////////////////

const double inf = 1e10;

bool cmp1(Point x,Point y){

if(isZero(x.y-y.y)){

return (x.x < y.x);

}else{

return (x.y < y.y);

}

}

double minDis(vector <Point> &p,int l,int r){

if(l==r){

return inf;

}

if(l==r-1){

return dis(p[l],p[r]);

}

int mid=(l+r)/2;

double ans = min(minDis(p,l,mid),minDis(p,mid+1,r));

vector <Point> k;

k.clear();

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

if(p[mid].x-p[i].x < ans+1e-6 && p[mid].x-p[i].x > -ans-1e-6){

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

}

}

sort(k.begin(),k.end(),cmp1);

const int len=7;

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

for(int j=i+1;j<k.size() && j<i+len ; j++){

ans = min(ans,dis(k[i],k[j]));

}

}

return ans;

}

int main(){

int n;

cin>>n;

vector <Point> a;

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

double x,y;

cin>>x>>y;

Point p(x,y);

a.push\_back(p);

}

sort(a.begin(),a.end(),cmp);

cout<<minDis(a,0,n-1)<<endl;

return 0;

a=convexHull(a);

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

cout<<a[i].x<<" "<<a[i].y<<endl;

}

}

///////////////////////NEW/////////////////////////////

#include <iostream>

#include <algorithm>

#include <vector>

#include <complex>

#include <cmath>

using namespace std;

const int MAX\_N=1000+10;

const double eps=1e-6;

typedef complex<double> point;

//Ax+By=C

struct line{

double A, B, C;

line(double A=0.0, double B=0.0, double C=0.0): A(A), B(B), C(C){}

};

bool is\_zero(double d){

return -eps<=d && d<=eps;

}

//product of q\_p1 p1\_p2

double dot(point q, point p1, point p2){

p2-=p1;

p1-=q;

return p1.real()\*p2.real()+p1.imag()\*p2.imag();

}

//product of q\_p1 q\_p2

double cross(

point q, point p1, point p2){

p1-=q; p2-=q;

return p1.real()\*p2.imag()-p1.imag()\*p2.real();

}

double Distance(const point& p, const point& q){

return sqrt((p.real()-q.real())\*(p.real()-q.real())+(p.imag()-q.imag())\*(p.imag()-q.imag()));

}

//Distance from P1\_P2 to Q

//if issegment is true P1\_P2 is segment

double LinePointDis(point p1, point p2, point q, int isSegment){

double ret=cross(p1,p2,q)/Distance(p1,p2);

if(isSegment){

if(dot(p1,p2,q)>0) return Distance(p2,q);

if(dot(p2,p1,q)>0) return Distance(p1,q);

}

if(ret<0) ret\*=-1;

return ret;

}

line point\_to\_line(const point& p1, const point& p2){

line ret;

ret.A=p2.imag()-p1.imag();

ret.B=p1.real()-p2.real();

ret.C=ret.A\*p1.real()+ret.B\*p1.imag();

return ret;

}

void line\_to\_point(const line& l, point& p1, point& p2){

if(is\_zero(l.A)){

p1=point(l.C/l.B, 0.0);

p2=point(l.C/l.B, 1.0);

}else{

p1=point((l.C-l.B\*0.0)/l.A, 0.0);

p2=point((l.C-l.B\*1.0)/l.A, 1.0);

}

}

bool line\_line\_inter(line l1, line l2, point& p){

double det=l1.A\*l2.B-l2.A\*l1.B;

if(is\_zero(det)){

return 0;

}else{

p=point( (l2.B\*l1.C-l1.B\*l2.C)/det, (l1.A\*l2.C-l2.A\*l1.C)/det );

return 1;

}

}

bool seg\_seg\_inter(const point& p1, const point& p2, const point& q1, const point& q2, point& p){

line l1=point\_to\_line(p1, p2);

line l2=point\_to\_line(q1, q2);

double det=l1.A\*l2.B-l2.A\*l1.B;

if(is\_zero(det)){

return 0;

}else{

p=point( (l2.B\*l1.C-l1.B\*l2.C)/det, (l1.A\*l2.C-l2.A\*l1.C)/det );

if(p.real()>=max(p1.real(), p2.real())+eps) return 0;

if(p.real()<=min(p1.real(), p2.real())-eps) return 0;

if(p.real()>=max(q1.real(), q2.real())+eps) return 0;

if(p.real()<=min(q1.real(), q2.real())-eps) return 0;

return 1;

}

}

bool find\_circle(point p1, point p2, point p3, point& o, double& r){

line l1=point\_to\_line(p1, p2);

swap(l1.A, l1.B);

l1.A\*=-1.0;

l1.C=(l1.A\*((p1.real()+p2.real())/2.0)+l1.B\*((p1.imag()+p2.imag())/2.0));

line l2=point\_to\_line(p1, p3);

swap(l2.A, l2.B);

l2.A\*=-1.0;

l2.C=(l2.A\*((p1.real()+p3.real())/2.0)+l2.B\*((p1.imag()+p3.imag())/2.0));

if(!line\_line\_inter(l1, l2, o)) return 0;

r=Distance(o, p1);

return 1;

}

vector<point> circle\_circle\_inter(point o1, double r1, point o2, double r2){

vector<point> ret;

double Dis=Distance(o1,o2);

if(Dis>=r1+r2+eps || Dis<=max(r1,r2)-min(r1,r2)-eps)

return ret;

o2-=o1;

r1/=abs(o2);

r2/=abs(o2);

double x=(1.0+r1\*r1-r2\*r2)/2.0;

double y=sqrt(max(0.0, r1\*r1-x\*x));

ret.push\_back(point(x,y));

if(!is\_zero(y))ret.push\_back(point(x,-y));

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

ret[i]\*=o2;

ret[i]+=o1;

}

return ret;

}

vector<point> line\_circle\_inter(point p1, point p2, point o, double r, bool isSegment){

vector<point> ret;

double Dis=LinePointDis(p1,p2,o,0);

if(Dis>=r+eps) return ret;

p1-=o; p2-=o;

line l1=point\_to\_line(p1,p2);

line l2(-l1.B, l1.A, 0.0);

point p;

line\_line\_inter(l1,l2,p);

if(is\_zero(abs(p))){

point ans1=p1/abs(p1)\*r;

point ans2=-ans1;

ret.push\_back(ans1+o);

ret.push\_back(ans2+o);

}else{

r/=abs(p);

double x=1.0;

double y=sqrt(max(0.0, r\*r-1.0));

ret.push\_back(point(x,y)\*p+o);

ret.push\_back(point(x,-y)\*p+o);

}

if(is\_zero(abs(ret[0]-ret[1])))

ret.pop\_back();

if(isSegment){

p1+=o; p2+=o;

vector<point> ans;

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

if(ret[i].real()>=min(p1.real(),p2.real())-eps && ret[i].real()<=max(p1.real(),p2.real())+eps

&& ret[i].imag()>=min(p1.imag(),p2.imag())-eps && ret[i].imag()<=max(p1.imag(),p2.imag())+eps)

ans.push\_back(ret[i]);

}

return ans;

}else{

return ret;

}

}

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

volume of part of sphere H is height fromt the buttom

R is the radius of sphere

counter clockwise rotation :

cost -sint \* x = x'

sint cost \* y = y'

**Stringstream:**

getline(cin,s);

stringstream IN(s);

int a;

while(IN>>a)

cout<<a<<" ";

**NIM:**

if xor daste ha = 0 nafare dovom

else nafare aval

farz knid xor daste ha = X

X ra ba har dasste XOR miknim agar un daste kam shod az hamun daste meghdari bar midarim ta barbaar ba adade jadid shavad

**Merge Sort + Inversion:**

ll n,a[MAX\_N],inv[MAX\_N];

pair<ll,ll>b[MAX\_N],tmp[MAX\_N];

//b[i].first = a[i], b[i].second = i

//inv[i] = inversions of index i

void mergeSort(ll left,ll right) {

if(left == right)

return;

ll mid = (left+right)/2;

mergeSort(left,mid);

mergeSort(mid+1,right);

ll k = 1,i = left,j = mid+1;

while(i<=mid && j <= right) {

if(b[i].first <= b[j].first) {

inv[b[i].second] += j-(mid+1);

tmp[k] = b[i];

i++;

k++;

}

else {

tmp[k] = b[j];

k++;

j++;

}

}

for(int q = i ; q <= mid ; q++,k++) {

inv[b[q].second] += (right-mid);

tmp[k] = b[q];

}

for(int q = j ; q <= right ; q++,k++)

tmp[k] = b[q];

k--;

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

b[left+q-1] = tmp[q];

}

int n,a[MAX\_N],fenv[MAX\_N];

//if MAX\_N is Maximum Number that exist we dont have to make between 1 to n

// numbers must be between 1 to n

void fen(ll x) {

while(x<MAX\_N)

{

fenv[x]++;

x += x & -x;

}

}

ll getsum(ll x) {

ll ret= 0;

while(x>0)

{

ret += fenv[x];

x -= x & -x;

}

return ret;

}

ll inversion() {

ll ret = 0;

for (int i=n-1; i>=0; i--) {

ret += getsum(a[i]-1);

fen(arr[i]);

}

return ret;

}

}

**Grid:**

for the triangle or hexigonal we put y diagonali

d(p, q) = (sin(plat) sin(qlat) + cos(plat) cos(qlat) cos(plong - qlong)(r)

distance of two point with coordinates of degree latitude and longtitude

latitude between 0 to 90(east -west line)(vasat 0 shomal o jonub 90)

langtitude between 0 to 180(north-south line)(greenwich 0 b samte jonub o shoma ta 180)

**BigDecimal BigInteger:**

BigDecimal bd = new BigDecimal("123.3123",mc);//can be negatvie

BigDecimal bd1 = new BigDecimal("151.66");

MathContext mc = new MathContext(212);//number of digits

bd.setScale(31,BigDecimal.ROUND\_DOWN);//number of digits of fraction

// all return a BigDecimal

bd.divide(bd1,55,BigDecimal.ROUND\_DOWN);//because it may have inifinty fraction we define number of digits of it

bd.add(bd1,mc);

bd.subtract(bd1,mc);

bd.multiply(bd1,mc);

bd.pow(n,mc);

bd.abs();

BigDecimal valueOf(double)

//add & multiply & subtract can have mathcontext so do constructor

BigInteger bi = new BigInteger("412");

BigInteger bi1 = new BigInteger("555"); //all returns a BigInteger

BigInteger valueOf(long)

bi.abs();

bi.add(bi1);

bi.multiply(bi1);

bi.subtract(bi1);

bi.divide(bi1);

bi.divide(bi1);

bi.remainder(bi1);

bi.modPow(bi1,mod);