Contents

[Relabel-to-front 2](#_Toc212113464)

[Hopcroft-Karp + Konig 2](#_Toc212113465)

[Data Flow 3](#_Toc212113466)

[Gaussian Elimination 4](#_Toc212113467)

[Degree (Graphic) Sequence 4](#_Toc212113468)

[Max Flow 4](#_Toc212113469)

[Cut Vertex 4](#_Toc212113470)

[Cut Edge 5](#_Toc212113471)

[Separable /\*Biconnected\*/ Component 5](#_Toc212113472)

[Euler Path 5](#_Toc212113473)

[Maximum weighted Bipartite Matching 5](#_Toc212113474)

[Unweighted General Matching – Blossom Shrinking 6](#_Toc212113475)

[Disjoint Set 7](#_Toc212113476)

[Longest Increasing Subsequence 7](#_Toc212113477)

[Range Minimum Query 7](#_Toc212113478)

[Knuth-Morris-Pratt Algorithm 7](#_Toc212113479)

[Aho-Corasick Algorithm 7](#_Toc212113480)

[2-D Interval Tree 8](#_Toc212113481)

[Tree Isomorphism 9](#_Toc212113482)

[Unrooted Tree DP 10](#_Toc212113483)

[Big Int 10](#_Toc212113484)

[STL 11](#_Toc212113485)

[Number Theory 12](#_Toc212113486)

[Mathematics 12](#_Toc212113487)

[Plane Geometry 13](#_Toc212113488)

[3-D Geometry 14](#_Toc212113489)

[Spherical Coordinate ( 14](#_Toc212113490)

[Computational Geometry 14](#_Toc212113491)

[Point and Line 14](#_Toc212113492)

[Closest Pair Problem 16](#_Toc212113493)

[Polygon Intersection 16](#_Toc212113494)

[Point inside polygon 18](#_Toc212113495)

[Distance between two polygons 18](#_Toc212113496)

[Distance between line segments 20](#_Toc212113497)

[Polygon Symmetry 21](#_Toc212113498)

[Rotation and Resize 21](#_Toc212113499)

[Polygon, eating d cm of its edges 23](#_Toc212113500)

[Smallest Enclosing Disk 24](#_Toc212113501)

[Complex Numbers 24](#_Toc212113502)

## Relabel-to-front

#include <iostream>

#include <string>

#include <list>

using namespace std;

#define MAXV 100

// one-based

int c[MAXV+1][MAXV+1]; // capacity

int f[MAXV+1][MAXV+1]; // flow

int h[MAXV+1]; // height

int e[MAXV+1]; // excess

list<int> L;

list<int> N[MAXV+1];

list<int>::iterator u;

list<int>::iterator r[MAXV+1];

int main() {

memset(c, 0, sizeof(c));

memset(f, 0, sizeof(f));

memset(h, 0, sizeof(h));

memset(e, 0, sizeof(e));

int n,m; // number of V and E

cin >> n >> m;

L.clear();

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

N[i].clear();

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

int x,y,f;

cin >> x >> y >> f;

N[x].insert(N[x].begin(), y);

N[y].insert(N[y].begin(), x);

c[x][y] += f;

}

int s, t; // source and sink

cin >> s >> t;

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

r[i] = N[i].begin();

if (i != s && i != t) L.insert(L.begin(), i);

if (c[s][i]) // init preflow

e[s] += (f[i][s] = -(f[s][i] = e[i] = c[s][i]));

}

h[s] = n;

u = L.begin();

while (u != L.end()) {

int x = \*u, old = h[\*u];

int v, F;

while (e[x] > 0) { // discharge

if (r[x] == N[x].end()) {

int z = n\*3;

for (list<int>::iterator i = N[x].begin(); i != N[x].end(); ++i)

if ((F=c[x][v=\*i]-f[x][\*i])>0)

z = h[v]<z?h[v]:z;

h[x] = z+1;

r[x] = N[x].begin();

} else {

v = \*r[x];

if ((F=c[x][v]-f[x][v]) > 0 && h[x] == h[v]+1) {

F = e[x]<F?e[x]:F;

f[x][v] += F; e[v] += F;

f[v][x] -= F; e[x] -= F;

if (e[x] <= 0) break;

}

++r[x];

}

}

if (h[x] > old) {

L.erase(u);

L.insert(L.begin(), x);

u = L.begin();

}

++u;

}

cout << "max flow = " << e[t] << endl;

}

## Hopcroft-Karp + Konig Bipartite matching+bipartite vertex cover

//needs vertices to be 1..n and 1..m

// UVa 11419 – Sam I Am

#include <iostream>

#include <cstring>

#include <queue>

using namespace std;

#define maxn 1001

#define maxm 1001

bool g[maxn][maxm],

vx[maxn], // matched nodes in L

cx[maxn], cy[maxm]; // vertex cover

long n,m,k,f, // n -> L, m -> R, k = |E|, f = flow

my[maxm]; // mate of R

bool aug(long x) {

if (vx[x])

return false;

vx[x] = 1;

for (int y = 1; y <= m; ++y)

if (g[x][y] && (!my[y] || aug(my[y]))) {

my[y] = x;

return true;

}

return false;

}

void vc() { // only Konig

for (int y = 1; y <= m; ++y)

if (my[y])

cx[my[y]] = 1;

queue<int> Q;

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

if (!cx[x])

Q.push(x);

while (!Q.empty()) {

int x = Q.front();

Q.pop();

for (int y = 1; y <= m; ++y)

if (g[x][y] && !cy[y] && my[y] != x) {

cy[y] = 1;

cx[my[y]] = 0;

Q.push(my[y]);

}

}

}

int main() {

while (scanf("%d%d%d", &n, &m, &k), n) {

memset(g, 0, sizeof g);

memset(my, 0, sizeof my);

memset(cx, 0, sizeof cx);

memset(cy, 0, sizeof cy);

int x,y;

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

scanf("%d%d", &x, &y);

g[x][y] = 1;

}

f = 0;

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

memset(vx, 0, sizeof vx);

if (aug(i)) ++f;

}

vc();

printf("%d", f);

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

if (cx[x])

printf(" r%d", x);

for (int y = 1; y <= m; ++y)

if (cy[y])

printf(" c%d", y);

printf("\n");

}

return 0;

}

## Data Flow

// problem:

// Given a graph, cost of each edge,

// capacity of edges (all equal)

// Flow cost defined as sum of each unit’s path cost

// e.g. A-B=3, A-C=2, A-D=1, B-D=4, C-D=5

// capacity = 10

// to flow 20 units from A to D

// first 10 units: directly from A to D

// next 10 units: A-B-D

// cost: A-D: 1 x 10 + A-B-D: 7 x 10 = 80

#include <iostream>

#include <string>

#include <queue>

#include <stack>

#include <algorithm>

using namespace std;

struct rec {

long long n;

long long k;

rec(long long n, long long k): n(n),k(k) {};

};

bool operator< (const rec& a, const rec& b) {

return a.k>b.k;

}

priority\_queue<rec> x[100][100];

int xx[100][100];

long long y[100],z[100];

bool w[100];

int main() {

int n,m;

while (cin>>n>>m) {

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

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

while (x[i][j].size())

x[i][j].pop();

memset(xx,0,sizeof(xx));

while (m--) {

long long a,b,c;

cin >> a >> b >> c;

a--; b--;

xx[a][b]=xx[b][a]=c;

}

long long d,k;

cin >> d >> k;

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

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

if (xx[i][j]>0)

x[i][j].push(rec(k,xx[i][j]));

try {

long long ans=0;

while (d>0) {

memset(y,0x3f,sizeof(y));

y[0]=0;

memset(w,0,sizeof(w));

w[0]=true;

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

bool b=true;

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

if (w[i]) {

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

if (x[i][j].size()>0) {

if (y[j]>y[i]+x[i][j].top().k) {

y[j]=y[i]+x[i][j].top().k;

z[j]=i;

w[j]=true;

b=false;

}

}

}

w[i]=false;

}

if (b) break;

}

if (y[n-1]==0x3f3f3f3f3f3f3f3fll)

throw 0;

stack<int> s;

long long l=d;

int a,b;

b=n-1;

while (b!=0) {

s.push(b);

a=z[b];

l=min(l,x[a][b].top().n);

b=a;

}

a=0;

while (!s.empty()) {

b=s.top();

s.pop();

rec r=x[a][b].top();

x[a][b].pop();

x[b][a].push(rec(l,-r.k));

r.n-=l;

if (r.n>0)

x[a][b].push(r);

ans+=r.k\*l;

a=b;

}

d-=l;

}

cout << ans << endl;

} catch (...) {

cout << "Impossible." << endl;

}

}

return 0;

}

## Gaussian Elimination

for (int i = nnmn; i < nn; ++i) {

int \*aa = a + (i >> 5);

int yy = 1 << (i & 31);

if (!get2(i)) { // Diagonal cell (i,i) not filled

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

if (get2(j)) { // Suitable row to xor with row i

xor(i,j,i);

break;

}

if (!get2(i)) continue; // Unsolvable row

}

for (int j = nnmn; j < nn; ++j)

if (j != i && get2(j))

xor(j,i,i);

}

## Degree (Graphic) Sequence

Let d\_1 >= d\_2 >= ... >= d\_n.

[Havel-Hakimi’s Theorem]

{d\_1, d\_2, ..., d\_n} is a degree sequence iff {d\_2-1, d\_3-1, ..., d\_{d1+1}-1, d\_{d1+2}, ..., d\_n} is a degree sequence.

Base case: {0, 0, 0, ..., 0} is a degree sequence.

## Max Flow

#include<iostream>

#include<algorithm>

using namespace std;

int n,m;

bool visit[101];

int map[101][101],f[101][101];

int s,t;

int dfs(int v,int k) {

if (v==t)

return k;

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

if (!visit[i]&&map[v][i]>0) {

visit[i]=true;

int delta =

dfs(i,min(k,map[v][i]));

if (delta>0) {

f[v][i]+=delta;

map[v][i]-=delta;

map[i][v]+=delta;

return delta;

}

}

return 0;

}

int main() {

int x,y,z,ans,tt,cntt=0;

cin>>n;

while (n!=0) {

ans = 0;

cin >> s >> t >> m;

memset(map,0,sizeof(map));

memset(visit,0,sizeof(visit));

memset(f,0,sizeof(f));

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

cin>>x>>y>>z;

map[x][y]+=z;

map[y][x]+=z;

}

while((tt=dfs(s,4000000))>0){

ans+=tt;

memset(visit,0,sizeof(visit));

visit[s]=true;

/\*

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

if (visit[i])

for(int j=2;j<=n;j++)

if (!visit[j]&&

f[i][j]>f[j][i]&&map[i][j]==0)

cout<<i<<" "<<j<<endl;

cout<<endl; \*/

}

cout << “Network “ << ++cntt <<”\nThe bandwidth is “ << ans <<”.\n”;

cin>>n;

}

return 0;

}

## Cut Vertex

// n = number of vertices

// a = adjacency list

// Cut vertices computed and stored in cut

int ttime,root;

vector<int> a[MAX];

int t[MAX],v[MAX];

int p[MAX],par[MAX],cut[MAX];

void dfs(int x) {

p[x] = t[x] = ttime++;

v[x] = 1;

int k = 0;

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

int y = a[x][i];

if (!v[y]) {

++k;

par[y] = x; dfs(y);

p[x] = min(p[x], p[y]);

if (x != root && p[y] >= t[x])

cut[x] = 1;

} else if (par[x] != y)

p[x] = min(t[y], p[x]);

}

if (x == root && k > 1)

cut[x] = 1;

}

int main() {

// construct graph

memset(v, 0, sizeof(v));

memset(cut, 0, sizeof(cut));

ttime = 0;

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

if (!v[i])

dfs(root = i);

}

## Cut Edge

// n = number of vertices

// a = adjacency list

// Cut vertices computed and stored in bridge

int ttime;

vector<pair<int,int> > bridge;

vector<int> a[MAX];

int t[MAX],v[MAX];

int p[MAX],par[MAX];

void dfs(int x) {

p[x] = t[x] = ttime++;

v[x] = 1;

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

int y = a[x][i];

if (!v[y]) {

par[y] = x; dfs(y);

p[x] = min(p[x], p[y]);

if (p[y] > t[x])

bridge.push\_back(

make\_pair(x,y));

} else if (par[x] != y)

p[x] = min(t[y], p[x]);

}

}

int main() {

// construct graph

memset(v, 0, sizeof(v));

ttime = 0;

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

if (!v[i])

dfs(i);

}

## Separable /\*Biconnected\*/ Component

// n = number of vertices

// a = adjacency list

// Vertices popped in line (\*\*\*)

// together with x forms a separable component

vector<int> a[MAX];

int t[MAX],v[MAX];

int p[MAX],par[MAX],ttime;

stack<int> S;

void dfs(int x) {

p[x] = t[x] = ttime++;

v[x] = 1;

S.push(x);

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

int y = a[x][i];

if (!v[y]) {

par[y] = x;

dfs(y);

p[x] = min(p[x], p[y]);

if (p[y] >= t[x]) {

while (S.top() != x)

S.pop(); // (\*\*\*)

}

}else if (par[x] != y)

p[x] = min(t[y], p[x]);

}

}

int main() {

// construct graph

memset(v, 0, sizeof(v));

ttime = 0;

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

if (!v[i])

dfs(i);

}

## Euler Path

void find\_path(int loc) {   
 int lv;   
      for (lv = 0; lv < nconn; lv++)     
          if (conn[loc][lv])     
            {     
              /\* delete edge \*/     
              /\* find path from new location \*/     
              find\_path(lv);     
            }     
          /\* add this node to the `end' of the path \*/

        path[plen++] = loc;     
    }

      for (lv = 0; lv < nconn; lv++)     
          if (deg[lv] % 2 == 1) break;     
      if (lv >= nconn)     
          for (lv = 0; lv < nconn; lv++)     
              if (deg[lv]) break;     
      /\* find the eulerian path \*/     
      find\_path(lv);       
      /\* the path is discovered in reverse order \*/     
      for (lv = plen-1; lv >= 0; lv--)     
          fprintf(fout, "%i\n", path[lv]+1);

## Maximum weighted Bipartite Matching

// n — number of vertices on each side

// 1 to n — A

// n + 1 to 2n — B

// Weights stored in g

#define r(x,y) (u[x]+u[y]-g[x][y])

#define MAX 100 // Max #vertices on each side

#define MAX2 MAX\*2+1

int g[MAX2][MAX2];

int m[MAX2], u[MAX2], d[MAX2], p[MAX2], v[MAX2];

int n, delta;

void D(int w, int x) { // DFS

if (v[x]) return;

v[x] = 1; p[x] = w;

if (x<=n) {

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

int fake\_slack = r(x,i) + delta;

if (!v[i] && fake\_slack < d[i])

d[i] = fake\_slack; p[i] = x;

}

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

if (!r(x,i) && m[x]!=i)

D(x,i);

}else{

if (!m[x]) throw x;

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

if (!r(i,x) && m[x]==i)

D(x,i);

}

}

void A(int x) { // augment

if (x==p[x]) return;

if (x>n)

m[m[x] = p[x]] = x;

A(p[x]);

}

int main() {

memset(m,0,sizeof(m));

memset(u,0,sizeof(u));

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

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

u[i] = g[i][j]>u[i] ? g[i][j] : u[i];

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

memset(v, 0, sizeof(v));

memset(d, 127, sizeof(d));

delta = 0;

try {

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

if (!m[i]) D(i,i);

while (1) {

int z = 0;

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

if (!v[i])

z = d[i]<d[z] ? i : z;

int real\_dz = d[z] - delta;

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

if (v[i])

u[i] -= real\_dz;

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

if (v[i])

u[i] += real\_dz;

delta = d[z];

D(p[z],z);

}

} catch (int x) {

A(x);

}

}

int S = 0;

for (int i=1; i<=n+n; ++i) S += u[i];

cout << "Total weight: " << S << endl;

// matching stored in m[]

}

## Unweighted General Matching – Blossom Shrinking

const int MAX = 100;

struct node {

int label, first, mate;

int l1, l2;

};

int n, m;

vector<int> edges[MAX+1];

node nodes[MAX+1];

queue<int> q;

int blossomCounter = -1;

void construct\_graph() {

cin >> n >> m;

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

edges[i].clear();

nodes[i].mate = 0;

}

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

int x, y;

cin >> x >> y;

edges[x].push\_back(y);

edges[y].push\_back(x);

}

}

void alternate(int v, int w) {

int t = nodes[v].mate;

nodes[v].mate = w;

if (nodes[t].mate != v)

return;

if (nodes[v].label <= n) {

nodes[t].mate = nodes[v].label;

alternate(nodes[v].label, t);

} else {

alternate(nodes[v].l1, nodes[v].l2);

alternate(nodes[v].l2, nodes[v].l1);

}

}

void blossomize(int v, int w, int flag) {

int r = nodes[v].first;

int join;

do {

nodes[r].label = flag;

if (!r) break;

r = nodes[nodes[nodes[r].mate].label].first;

} while (1);

r = nodes[w].first;

do {

if (nodes[r].label == flag || !r) { join = r; break; }

r = nodes[nodes[nodes[r].mate].label].first;

} while (1);

int temp[2] = {v, w};

for (int i=0; i<2; ++i) {

r = nodes[temp[i]].first;

while (r != join) {

nodes[r].label = n+1;

nodes[r].l1 = v;

nodes[r].l2 = w;

nodes[r].first = join;

q.push(r);

r = nodes[nodes[nodes[r].mate].label].first;

}

}

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

if (nodes[nodes[i].first].label >= 0)

nodes[i].first = join;

}

bool augment(int u) {

cout << "augment " << u;

while (!q.empty()) q.pop();

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

nodes[i].label = -1;

nodes[u].label = 0;

nodes[u].first = 0;

q.push(u);

while (!q.empty()) {

int v = q.front(); q.pop();

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

int w = edges[v][i];

if (!nodes[w].mate && w!=u) {

nodes[w].mate = v;

alternate(v, w);

return true;

}

if (nodes[w].label >= 0)

blossomize(v, w, --blossomCounter);

else {

int x = nodes[w].mate;

if (nodes[x].label < 0) {

nodes[x].label = v;

nodes[x].first = w;

q.push(x);

}}}}

return false;

}

int main() {

construct\_graph();

int matching\_size = 0;

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

if (!nodes[i].mate)

if (augment(i))

matching\_size++;

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

if (nodes[i].mate && i<nodes[i].mate)

cout << " " << i << " " << nodes[i].mate << endl;

return 0;

}

## Disjoint Set

#define MAX 105

int head[MAX];

int find(int i){

return head[i] = (i == head[i]? i : find(head[i]));

}

int merge(int i,int j){

head[find(i)] = find(j);

}

## Longest Increasing Subsequence

#include <cstdio>  
#include <vector>  
using namespace std;  
  
template<typename T> vector<T> find\_lis(vector<T> &a)  
{  
    vector<int> b, p(a.size());  
    int u, v;  
   
    if (a.size() < 1) return b;  
   
    b.push\_back(0);  
   
    for (size\_t i = 1; i < a.size(); i++) {  
        if (a[b.back()] < a[i]) {  
            p[i] = b.back();  
            b.push\_back(i);  
            continue;  
        }  
   
        for (u = 0, v = b.size()-1; u < v;) {  
            int c = (u + v) / 2;  
            if (a[b[c]] < a[i]) u=c+1; else v=c;  
        }  
   
        if (a[i] < a[b[u]]) {  
            if (u > 0) p[i] = b[u-1];  
            b[u] = i;  
        }      
    }  
   
    for (u = b.size(), v = b.back(); u--; v = p[v]) b[u] = v;  
    return b;  
}  
  
/\*  
  Usage: lis = find\_lis(seq)  
  Result: "Sequence" = seq[lis[i]], "Size" = lis.size()  
\*/  
   
int main()  
{  
    vector<int> seq;  
    int t;  
    while (scanf("%d", &t) != EOF) seq.push\_back(t);  
    vector<int> lis = find\_lis(seq);  
  
    printf("%d\n-\n", lis.size());  
    for (size\_t i = 0; i < lis.size(); i++)  
        printf("%d\n", seq[lis[i]]);  
   
    return 0;  
}

## Range Minimum Query

// O(nlog(n)) pre-processing time, O(1) query time

#include <iostream>

#include <algorithm>

using namespace std;

const int MAXN = 1000010;

int a[MAXN];

int m[MAXN][20];

int l2[MAXN];

int main(){

int p = 0;

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

if ((1 << (p+1)) == i) l2[i] = ++p;

else l2[i] = p;

int n, w, q, l, u, i1, i2;

while (scanf("%d", &n), n){

for (int i = 0; i < n; ++i){ scanf("%d", a+i); m[i][0] = i; }

w = 1;

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

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

if (a[i1 = m[j][i-1]] <= a[i2 = m[j+w][i-1]]) m[j][i] = i1;

else m[j][i] = i2;

w += w;

}

scanf("%d", &q);

while (q--){

scanf("%d%d", &l, &u);

w = l2[u - l + 1];

if (a[i1 = m[l][w]] <= a[i2 = m[u-(1<<w)+1][w]]) cout << i1 << '\n';

else cout << i2 << '\n';

}

}

}

## Knuth-Morris-Pratt Algorithm

// Strings are assumed to be 1-based

// P[1..m] - pattern

// T[1..n] - text

#define MAX 1000

int pi[MAX]; // MAX >= m

int m,n;

void COMPUTE\_PREFIX\_FUNCTION(const char \* P) {

m = strlen(P+1);

pi[1] = 0;

int k = 0;

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

while (k && P[k+1]!=P[i])

k = pi[k];

if (P[k+1]==P[i]) ++k;

pi[i] = k;

}

}

void KMP\_MATCHER(const char \* T, const char \* P) {

n = strlen(T+1);

int q = 0;

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

while (q && P[q+1]!=T[i])

q = pi[q];

if (P[q+1]==T[i]) ++q;

if (q==m) {

cout << i-m+1 << endl; // index of P occur in T

q = pi[q];

}

}

}

## Aho-Corasick Algorithm

#include <iostream>

#include <algorithm>

#include <utility>

#include <list>

#include <cstring>

#include <queue>

using namespace std;

const char ter\_char = (char) 52;

struct node{

node\* v[52]; // a-z A-Z

node \* link;

int a[52]; // all valid links

int n; // no. of elements in a

list<int> val; // the pattern that matched

node(){

memset(v,0,sizeof(v));

link = 0; n = 0;

}

};

typedef pair<node \*, node \*> pr1;

// parent node, cur node

typedef pair<char,pr1> pr;

// char of the edge, edge

typedef list<int>::iterator si;

node \* root;

bool vis[1005] = {0};

// true if the pattern is matched

void insert(char \* s, int k){ // pattern and pattern index

node \* p = root;

for (int i = 0; s[i]!=ter\_char; ++i){

if (!(p->v[(int)s[i]])){

p->v[(int)s[i]] = new node;

p->a[(p->n)++] = s[i];

}

p = p->v[(int)s[i]];

}

p->val.push\_back(k);

}

void removeAll(node \* cur){

for (int i = 0; i < cur->n; ++i)

removeAll(cur->v[cur->a[i]]);

delete cur;

}

void createLink(){

queue<pr> q;

for (int i = 0; i < 52; ++i)

if (!(root->v[i]))

root->v[i] = root;

else q.push(pr(i,pr1(root,root->v[i])));

while (!q.empty()){

pr tt = q.front();

node \* prev = tt.second.first;

node \* cur = tt.second.second;

int c = tt.first;

int & n = cur->n;

q.pop();

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

q.push(pr(cur->a[i],pr1(cur,cur->v[cur->a[i]])));

if (prev == root){

cur->link = root;

}

else{

prev = prev->link;

while (!(prev->v[c]))

prev = prev->link;

prev = prev->v[c];

cur->link = prev;

list<int> temp = prev->val;

cur->val.merge(temp);

}

}

}

void match(char \* s){

memset(vis,0,sizeof(vis));

node \* p = root;

for (int i = 0; s[i]!=ter\_char; ++i){

while (!(p->v[(int)s[i]])) p = p->link;

p = p->v[(int)s[i]];

for (si i = p->val.begin(); i != p->val.end(); ++i)

vis[\*i] = true;

}

}

/\*\*\* USE METHOD \*\*\*

root = new node();

Map all character in {Pi} from a-z A-Z to 0..25 26..51

and terminating null character to 52

for each patterm Pi, insert(P,i);

createLink();

match(T); where T is the target string

result stored in vis

removeAll(root);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

char s[100010];

char t[1010][1010];

int main(){

long tt,i,j,n,z;

cin>>tt;

gets(s);

for (z=0; z<tt; ++z){

root=new node();

gets(s);

j=0;

while (s[j]){

if (s[j]>='a')

s[j]= s[j]-'a';

else

s[j]= s[j]+26-'A';

if (!s[j+1]){

s[j+1] = ter\_char;

break;

}

j+=1;

}

cin>>n;

gets(t[0]);

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

gets(t[i]);

j=0;

while (t[i][j]){

if (t[i][j]>='a')

t[i][j]= t[i][j]-'a';

else

t[i][j]= t[i][j]+26-'A';

if (!t[i][j+1]){

t[i][j+1]= ter\_char;

break;

}

j+=1;

}

insert(t[i],i);

}

createLink();

match(s);

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

if (vis[i])

cout<<'y'<<endl;

else

cout<<'n'<<endl;

removeAll(root);

}

return 0;

}

## 2-D Interval Tree

#include<iostream>

using namespace std;

struct ynode{

int num;

ynode \*left,\*right;

ynode():num(0),left(NULL),right(NULL){}

};

struct xnode{

ynode \*y;

xnode \*left,\*right;

xnode():left(NULL),right(NULL),y(NULL){}

};

int maxy;

void inserty(ynode\* &now,int left,int right, int targety, int k){

if(now==NULL)

now=new ynode;

now->num+=k;

if(left==right) return;

int m=(left+right)/2;

if(targety<=m)

inserty(now->left,left,m,targety,k);

else

inserty(now->right,m+1,right,targety,k);

}

void insertx(xnode\* &now,int left, int right, int targetx, int targety, int k){

if(now==NULL)

now=new xnode;

inserty(now->y,0,maxy,targety,k);

if(left==right) return;

int m=(left+right)/2;

if(targetx<=m)

insertx(now->left,left,m,targetx,targety,k);

else

insertx(now->right,m+1,right,targetx,targety,k);

}

int queryy(ynode\* now,int left, int right,int targety){

if(now==NULL) return 0;

if(left==right) return now->num;

int m=(left+right)/2;

if(targety<=m)

return queryy(now->left,left,m,targety);

else {

int c=0;

if(now->left!=NULL)

c=now->left->num;

return c+queryy(now->right,m+1,right,targety);

}

}

int queryx(xnode\* now,int left, int right,int targetx, int targety){

if(now==NULL)

return 0;

if(left==right) return; queryy(now->y,0,maxy,targety);

int m=(left+right)/2;

if(targetx<=m)

return queryx(now->left,left,m,targetx,targety);

else {

int c=0;

if(now->left!=NULL)

c=queryy(now->left->y,0,maxy,targety);

return c+queryx(now->right,m+1,right,targetx,targety);

}

}

int main(){

int n,k,x1,x2,y1,y2;

cin>>n;

xnode \*root;

while(n!=3){

if(n==0) {root=NULL;cin>>maxy;maxy++;}

if(n==1) {cin>>x1>>y1>>k;insertx(root,0,maxy,x1+1,y1+1,k);}

if(n==2) {

cin>>x1>>y1>>x2>>y2;

int ans=queryx(root,0,maxy,x2+1,y2+1)-queryx(root,0,maxy,x1,y2+1)-queryx(root,0,maxy,x2+1,y1)+queryx(root,0,maxy,x1,y1);

cout<<ans<<endl;}

cin>>n;

}

return 0;

}

// Likyau’s version

#include <iostream>

using namespace std;

const long dim=2,MAX=(1<<28),MIN=-(1<<28);

long e[dim][1025][2];

long vmin[1025][1025],vmax[1025][1025];

long n,m,p,q,r,s,maxi,mini,size[dim];

long con[dim];

long list[2][1002];

bool bottom;

struct node{

long l[dim];

};

void build(long lv,long a,long b,long k){

if (k>size[lv])

size[lv]=k;

e[lv][k][0]=a;

e[lv][k][1]=b;

if (a!=b){

build(lv,a,(a+b) /2,2\*k+1);

build(lv,(a+b) /2+1,b,2\*k+2);

}

}

void retrieve(long lv,long a,long b,long k){

long x=(e[lv][k][0]+e[lv][k][1]) / 2;

if (a<=b){

if ((a==e[lv][k][0]) && (b==e[lv][k][1]) ){

list[lv][con[lv]]=k;

con[lv]+=1;

}

else if (b<=x){

retrieve(lv,a,b,2\*k+1);

}

else if (a>x){

retrieve(lv,a,b,2\*k+2);

}

else{

retrieve(lv,a,x,2\*k+1);

retrieve(lv,x+1,b,2\*k+2);

}

}

}

void retrieve\_batch(long a,long b,long c,long d){

con[0]=0;

con[1]=0;

retrieve(0,a,c,0);

retrieve(1,b,d,0);

maxi=MIN;

mini=MAX;

for (long i=0; i<con[0]; ++i)

for (long j=0; j<con[1]; ++j){

long x=list[0][i];

long y=list[1][j];

if (vmax[x][y]>maxi)

maxi=vmax[x][y];

if (vmin[x][y]<mini)

mini=vmin[x][y];

}

}

void compare(node tr){

long temp=MIN;

long x=tr.l[0];

long y=tr.l[1];

if (e[0][x][0]!=e[0][x][1]){

if (vmax[2\*x+1][y]>temp)

temp=vmax[2\*x+1][y];

if (vmax[2\*x+2][y]>temp)

temp=vmax[2\*x+2][y];

}

if (e[1][y][0]!=e[1][y][1]){

if (vmax[x][2\*y+1]>temp)

temp=vmax[x][2\*y+1];

if (vmax[x][2\*y+2]>temp)

temp=vmax[x][2\*y+2];

}

vmax[x][y]=temp;

temp=MAX;

if (e[0][x][0]!=e[0][x][1]){

if (vmin[2\*x+1][y]<temp)

temp=vmin[2\*x+1][y];

if (vmin[2\*x+2][y]<temp)

temp=vmin[2\*x+2][y];

}

if (e[1][y][0]!=e[1][y][1]){

if (vmin[x][2\*y+1]<temp)

temp=vmin[x][2\*y+1];

if (vmin[x][2\*y+2]<temp)

temp=vmin[x][2\*y+2];

}

vmin[x][y]=temp;

}

void insert(long lv,node pt,node tr,long va){

long k=tr.l[lv],x;

long i=tr.l[0]; long j=tr.l[1];

node temp=tr;

if (e[lv][k][0]==e[lv][k][1] && lv==dim-1 && bottom) {//when bottom is reached

vmax[tr.l[0]][tr.l[1]]=va;

vmin[tr.l[0]][tr.l[1]]=va;

bottom=false;

}

else{

if (e[lv][k][0]!=e[lv][k][1]){

x=(e[lv][k][0]+e[lv][k][1]) / 2;

if (pt.l[lv]<=x)

temp.l[lv]=2\*k+1;

else

temp.l[lv]=2\*k+2;

insert(lv,pt,temp,va);

}

if (lv!=dim-1)

insert(lv+1,pt,tr,va); //for loop if more lvl

compare(tr);

}

}

int main(){

char c;

long i,j;

node t1,t2;

t2.l[0]=0; t2.l[1]=0;

cin>>n>>n;

size[0]=0; size[1]=0;

build(0,1,n,0); build(1,1,n,0);

for (i=0; i<=size[0]; ++i)

for (j=0; j<=size[1]; ++j){

vmin[i][j]=MAX;

vmax[i][j]=MIN;

}

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

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

cin>>r;

t1.l[0]=i; t1.l[1]=j;

bottom=true;

insert(0,t1,t2,r);

}

cin>>m;

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

cin>>c;

if (c=='c'){

cin>>p>>q>>r;

t1.l[0]=p; t1.l[1]=q;

bottom=true;

insert(0,t1,t2,r);

}

else{

cin>>p>>q>>r>>s;

retrieve\_batch(p,q,r,s);

cout<<maxi<<" "<<mini<<endl;

}

}

return 0;

}

## Tree Isomorphism

#include <iostream>

#include <string>

#include <map>

#include <set>

using namespace std;

const int MAX\_N = 10010;

int e[2][MAX\_N][MAX\_N];

int ec[2][MAX\_N];

int tec[2][MAX\_N];

int temp[2][MAX\_N];

int p[2][MAX\_N];

int d[2][MAX\_N][MAX\_N];

int dc[2][MAX\_N];

int l[2][MAX\_N];

int m[2], n[2], height[2], root[2][2], rc[2];

void find\_depth(int root, int depth, int tree, int parent){

p[tree][root] = parent;

d[tree][depth][dc[tree][depth]++] = root;

height[tree] >?= depth;

++depth;

for (int i = 0; i < ec[tree][root]; ++i)

if (e[tree][root][i] != parent)

find\_depth(e[tree][root][i], depth, tree, root);

}

int main(){

int z, u, v;

cin >> z;

while (z--){

// read input

for (int i = 0; i < 2; ++i){

cin >> n[i] >> m[i];

memset(ec[i], 0, 4\*n[i]);

while (m[i]--){

cin >> u >> v;

e[i][u][ec[i][u]++] = v;

e[i][v][ec[i][v]++] = u;

}

}

if (n[0] != n[1]) goto fail;

if (n[0] == 1) goto pass;

// find centre(s)

for (int i = 0; i < 2; ++i){

int tc[2]; tc[0] = 0;

int nodes = n[0];

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

tec[i][j] = ec[i][j];

if (ec[i][j] == 1){

temp[0][tc[0]++] = j;

--nodes;

}

}

int row = 0;

while (nodes > 0){

tc[1 - row] = 0;

for (int j = 0; j < tc[row]; ++j)

for (int k = 0; k < ec[i][temp[row][j]]; ++k)

if (--tec[i][e[i][temp[row][j]][k]] == 1){

temp[1 - row][tc[1 - row]++] = e[i][temp[row][j]][k];

--nodes;

}

row = 1 - row;

}

root[i][0] = temp[row][0];

rc[i] = 1;

if (tc[row] == 2){

root[i][1] = temp[row][1];

rc[i] = 2;

}

}

if (rc[0] != rc[1]) goto fail;

// find depth

int r[2];

r[0] = root[0][0];

height[0] = 0;

memset(dc[0], 0, 4\*n[0]);

find\_depth(r[0], 0, 0, -1);

for (int i = 0; i < rc[1]; ++i){

r[1] = root[1][i];

height[1] = 0;

memset(dc[1], 0, 4\*n[1]);

find\_depth(r[1], 0, 1, -1);

if (height[0] != height[1]) continue;

// label leaves

for (int j = 0; j < 2; ++j)

for (int k = 0; k < dc[j][height[j]]; ++k)

l[j][d[j][height[0]][k]] = 0;

// label subtrees

for (int j = height[0] - 1; j >= 0; --j){

map<multiset<int>, int> labels;

for (int k = 0; k < 2; ++k){

for (int kk = 0; kk < dc[k][j]; ++kk){

int node = d[k][j][kk];

multiset<int> ss;

for (int kkk = 0; kkk < ec[k][node]; ++kkk)

if (e[k][node][kkk] != p[k][node])

ss.insert(l[k][e[k][node][kkk]]);

labels[ss] = 0;

}

}

int count = 0;

for (map<multiset<int>, int>::iterator itr = labels.begin();

itr != labels.end(); ++itr)

itr->second = count++;

for (int k = 0; k < 2; ++k){

for (int kk = 0; kk < dc[k][j]; ++kk){

int node = d[k][j][kk];

multiset<int> ss;

for (int kkk = 0; kkk < ec[k][node]; ++kkk)

if (e[k][node][kkk] != p[k][node])

ss.insert(l[k][e[k][node][kkk]]);

l[k][node] = labels[ss];

}

}

}

if (l[0][r[0]] == l[1][r[1]]) goto pass;

}

fail:

cout << "Different patterns\n";

continue;

pass:

cout << "Same pattern\n";

}

}

## Unrooted Tree DP

// NOI 2003 Hookey

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

#define maxn 200001

struct node{

int to;

long long cost;

node \*next;

node(int a,long long b,node \*c):to(a),cost(b),next(c){}

};

node \*path[maxn];

long long dp[maxn][3];

long long ans;

int maxroot[maxn];

int m,n;

bool change(int u, int k, long long vc)

{

if(vc>dp[u][k])

{

for(int i=2;i>k;i--)

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

dp[u][k]=vc;

return true;

}

return false;

}

void dfs(int f,int u)

{

for(node \*cur=path[u];cur!=NULL;cur=cur->next)

if(cur->to != f)

{

int v=cur->to;

dfs(u,v);

long long vc=cur->cost+dp[v][0];

if(change(u,0,vc)) maxroot[u]=v;

else if(change(u,1,vc));

else dp[u][2]=max(dp[u][2],vc);

}

}

void dfs2(int f,int u,long long vc)

{

if(maxroot[f]==u)

vc+=dp[f][1];

else

vc+=dp[f][0];

if(change(u,0,vc)) maxroot[u]=f;

else if(change(u,1,vc));

else dp[u][2]=max(dp[u][2],vc);

for(node \*cur=path[u];cur!=NULL;cur=cur->next)

if(cur->to!=f)

dfs2(u,cur->to,cur->cost);

}

int main()

{

int x,y,z;

cin>>n>>m;

memset(dp,0,sizeof(dp));

for(int i=1;i<=n;i++) path[i]=NULL;

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

{

cin>>x>>y>>z;

path[x]=new node(y,z,path[x]);

path[y]=new node(x,z,path[y]);

}

dfs(0,1);

for(node \*cur=path[1];cur!=NULL;cur=cur->next)

dfs2(1,cur->to,cur->cost);

ans=0;

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

ans=max(ans,dp[i][0]+2\*dp[i][1]+dp[i][2]);

cout<<ans<<endl;

return 0;

}

## Big Int //cmath, cstring

const int base=1000000000;

const int width=9;

const int size=100;

typedef struct BigInt {

int len;

int d[size];

BigInt(int n=0) {

len=1;

d[0]=n%base;

if (n>=base) {

d[1]=n/base;

len=2;

}

}

BigInt(string s) {

len=0;

int i=int(s.length());

while (i>0) {

if (i<width) {

d[len++]=atoi(s.substr(0,i).c\_str());

i=0;

} else {

i-=width;

d[len++] = atoi(s.substr(i,width).c\_str());

}

}

while (len>1 && d[len-1]==0)

len--;

} // End of BigInt(string)

};

BigInt operator+ (const BigInt& a, const BigInt& b) {

BigInt c;

int i,k=0;

for (i=0; i<a.len || i<b.len ||

k; i++) {

c.d[i]=k+(i<a.len? a.d[i]:0)+

(i<b.len? b.d[i]:0);

k=c.d[i]/base;

c.d[i]%=base;

}

c.len=i;

return c;

}

BigInt operator- (const BigInt& a, const BigInt& b) {

BigInt c;

int i,k=0;

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

c.d[i] = a.d[i] -

(i<b.len? b.d[i]:0)-k;

if (c.d[i]<0){

c.d[i]+=base;

k=1;

} else k=0;

}

c.len=a.len;

while (c.len>1 &&

c.d[c.len-1]==0)

c.len--;

return c;

}

BigInt operator\* (const BigInt& a, const BigInt& b){

if (a.len==1 && a.d[0]==0 ||

b.len==1 && b.d[0]==0)

return 0;

BigInt c;

memset(c.d,0,sizeof(c.d));

int i,j,k=0;

long long l;

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

for (j=0; j<b.len|| k;j++){

l=static\_cast<long long>(a.d[i])\*

(j<b.len? b.d[j]:0)+k+c.d[i+j];

c.d[i+j]=l%base;

k=l/base;

}

}

c.len=i+j-1;

return c;

}

bool operator< (const BigInt &a,const BigInt &b){

if (a.len!=b.len)

return a.len<b.len;

for (int i = a.len-1; i >=0; --i)

if (a.d[i]!=b.d[i])

return (a.d[i]<b.d[i]);

return false;

}

bool operator== (const BigInt &a,const BigInt &b){

if (a.len!=b.len)

return false;

for (int i = a.len-1; i >=0; --i)

if (a.d[i]!=b.d[i])

return false;

return true;

}

// need <,+,-,\* operator

// c = a/b, k = a%b

void bdiv(const BigInt& a, const BigInt& b, BigInt& c, BigInt& k) {

int i,p,q,m;

k=0;

BigInt l,t;

for (i=a.len-1; i>=0; i--){

l=k\*base+a.d[i];

p=0; q=base-1;

while (p<q) {

m=(p+q+1)/2;

t=b\*m;

if (l<t) q=m-1;

else p=m;

}

c.d[i]=p;

k=l-b\*p;

}

c.len=a.len;

while (c.len>1 &&

c.d[c.len-1]==0)

c.len--;

}

// need <,+,-,\* operator

// return square root of s

BigInt Sqrt(string s){

if (s=="0") return 0;

if (s.length()%2)

s='0'+s;

BigInt ans(0),A(0);

for (unsigned i=0; i<s.length();

i+=2){

A = A\*100+

((s[i]-'0')\*10+(s[i+1]-'0'));

BigInt F=ans\*20,M(0),N;

int k;

for (k=0; k<10; k++) {

N=M+F+(2\*k+1);

if (N>A) break;

M=N;

}

A=A-M;

ans=ans\*10+k;

}

return ans;

}

BigInt powx(int x,int y){

if (y==0)

return BigInt(1);

if (y==1)

return BigInt(x);

BigInt temp = powx(x,y/2);

if (y%2==1)

return temp\*temp\*BigInt(x);

return temp\*temp;

}

string output(BigInt b){ //iomanip,sstream

ostringstream oss;

oss << b.d[b.len-1];

for (int i = b.len-2;i>=0; --i)

oss <<setw(width)<<setfill('0') << b.d[i];

return oss.str();

}

## STL

Assume S = start, E = end,

M = middle, ITR = iterator

### List functions

// move all elements from x to before pos

void splice(ITR pos,list& x);

void splice(ITR pos, list& x, ITR p);

void splice(ITR pos,list& x,ITR S,ITR E);

void merge(list&);

void sort();

### Set functions

//pre: set1, set2 are sorted

// result is big enough

set\_difference

set\_intersection

set\_union

(ITR S1, ITR E1, ITR S2, ITR E2,

ITR result[, comp])

### General STL

lower\_bound(ITR S, ITR E, val,[comp]);

next\_permutation(ITR S,ITR E,[comp]);

// Rearrange array, [S,M) are smallest elements and sorted

partial\_sort( ITR S, ITR M, ITR E);

// Post: [S, nth) <= nth <= [nth+1,E)

nth\_element(ITR S, ITR nth, ITR E);

sort( ITR S,ITR E, [comp]);

stable\_sort( ITR S, ITR E, [comp]);

random\_shuffle(ITR S, ITR E);

### I/O Formatting

cin.get() <=> getchar()

cin.getline(buf,size,delim)

cin.peek()

cin.unget()

cout.setf(ios::fixed);

setprecision(n)

setw(n), setfill(c)

setiosflags(flag)

ios::fixed, ios::left, ios::right

ios::scientific

## Number Theory

### Extended Euclidean Algorithm

int EX\_EUCLID(a,b,&x,&y) {

// returns d = gcd(a,b) = ax + by

if (!b) {

x=1; y=0; return a;

}

int d,x2,y2;

d = EX\_EUCLID(b, a%b, x2, y2);

x = y2; y = x2 - a / b \* y2;

return d;

}

### Linear congruence *ax* ≡ *b* (mod *n*)

• Requires Extended Euclidean Algorithm.

• Prints solutions of *ax* ≡ *b* (mod *n*) less than *n*

d = EX\_EUCLID(a,n,x,y);

if (d%b) { no solution; }

else {

int x0 = b/d\*x%n;

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

output (x0 + n/d\*i)%n;

}

### Modular Inverse

// pre: a must be in range 1..m-1

// post: returns i in range 1..m-1 such that i\*a = 1(mod m)

long modinv(const long &a, const long &m) {

// modular inverse

long j = 1, i = 0u, b = m, c = a, x, y;

while (c != 0u) {

x = b / c;

y = b - x \* c;

b = c;

c = y;

y = j;

j = i - j \* x;

i = y; }

if (i < 0u) i += m;

return i;

}

## Mathematics

### Formulae 12 + 22 + ... + *n*2 =

13 + 23 + ... + *n*3 = 

****

****

****

****

For random variable X>0,



### Taylor Expansion



sin *x* = 

cos *x* = 

e*x* = 

### Lagrange Interpolation

****

****

### Day of Week

// caution: modifies d and y

(d+=m<3?y--:y-2,23\*m/9+d+4+

y/4-y/100+y/400)%7

### Combinatorics

** =**

No. of ways to choose ***r*** objects from ***n*** choices where each choice can be chosen any times (≥0) and where the order does not matter

OR

Number of ways to partition ***r*** objects into ***n*** partitions, which can possibly be empty

### Stirling Number of the Second Kind

Number of ways a set of N elements can be partitioned in to K non-empty sets.

Formula:



Recurrences:



### Bell Numbers

Number of way N objects can be partitioned into non-empty groups.

1, 1, 2, 5, 15, …

Formula:

****

Bell’s Triangle for generating Bell Number in O(n2)

**1**

**1** 2

**2** 3 5

**5** 7 10 15

**15** 20 27 37 52

Catalan number – the number of ways n edges can be arranged as a binary tree



## Plane Geometry

### Area of Triangle

ha is height of a

* + base \* height / 2
  + a\*b\*sinC / 2
  + sqrt(s\*(s-a)\*(s-b)\*(s-c))

where s = (a+b+c)/2



### Area of Ellipse

PI \* semimajor \* semiminor

### Area of Sector

r \* r \* angle\_radian / 2

### Vieta’s Formulae

*f*(*x*) = *anxn*+*an-1xn-1*+...+*a1x*+*a0*



****

### Centers

Inradius *r*, circumradius *R*, area ∆, semiperimeter *s*

R=

= 

= 

= 

*r* = 

= ∆/*s*

= 4*R* sin(*A*/2) sin(*B*/2) sin(*C*/2)

Incenter: (*xaa*+*xbb*+*xcc*) / (*a*+*b*+*c*)

Circumcenter:

(*xa*sin2*A*+*xb*sin2*B*+*xc*sin2*C*) /

(sin2*A*+sin2*B*+sin2*C*)

Orthocenter:

(*xa*tan*A*+ *xb*tan*B*+*xc*tan*C*) /

(tan*A*+tan*B*+tan*C*)

Excenter:

(-*xaa*+ *xbb*+*xcc*) / (-*a*+*b*+*c*)

Centroid:

(*xa*+*xb*+*xc*) / 3

Distance between incenter and circumcenter =

### Trigonometry

sin (*A* + *B*) = sin *A* cos *B* + cos *A* sin *B*

cos (*A* + *B*) = cos *A* cos *B* – sin *A* sin *B*

tan (*A* + *B*) = 

sin 2*A* = 2 sin *A* cos *A*

cos 2*A* = 2cos2*A*-1

= 1 – 2sin2*A*

= cos2*A* – sin2*A*

tan 2*A* = 

sin2*A* = , cos2*A* = 

|  |  |  |  |
| --- | --- | --- | --- |
|  | **-x** | **π-x** | **π/2+x** |
| **sin** | -sin x | sin x | cos x |
| **cos** | cos x | -cos x | -sin x |
| **tan** | -tan x | -tan x | -cot x |

## 3-D Geometry

### Rotations in Space

About x-axis, from Y to Z



About y-axis, from Z to X



About z-axis, from X to Y



### Rotate the point to (k,0,0)



where 

### Spherical Coordinate (r,Φ,θ)





### Plane (ax+by+cz+d=0)





### Great-circle Distance

d(p,q) = r cos-1(sinplatsinqlat+

cosplatcosqlatcos(plong-qlong))

## Computational Geometry

Point and Line

const double E=1e-8;

struct **pt**{

double x,y;

};

struct **line**{

double a,b,c;

};

line **makeline**(pt a,pt b){

line l;

l.a=a.y-b.y;

l.b=b.x-a.x;

l.c=-1\*l.b\*a.y-l.a\*a.x;

return l;

}

bool **inlinerange**(

const pt& a,const pt& b,const pt& c)

{

if (min(a.x,b.x)-E<c.x &&

c.x<max(a.x,b.x)+E &&

min(a.y,b.y)-E<c.y &&

c.y<max(a.y,b.y)+E )

return true;

else

return false;

}

bool **ontheline**(line l,pt p){

return (fabs(l.a\*p.x+l.b\*p.y+l.c)<E);

}

bool InsidePolygon(const vector<pt> polygon,pt p)

{

int N = polygon.size();

int counter = 0;

int i;

double xinters;

pt p1,p2;

p1 = polygon[0];

for (i=1;i<=N;i++) {

p2 = polygon[i % N];

if (p.y > min(p1.y,p2.y)) {

if (p.y <= max(p1.y,p2.y)) {

if (p.x <= max(p1.x,p2.x)) {

if (p1.y != p2.y) {

xinters = (p.y-p1.y)\*(p2.x-p1.x)/(p2.y-p1.y)+p1.x;

if (p1.x == p2.x || p.x <= xinters)

counter++;

}

}

}

}

p1 = p2;

}

if (counter % 2 == 0)

return(0);

else

return(1);

}

bool **parallel**(const pt& a,const pt& b,const pt& c,const pt& d)

{

if ((b.x==a.x)&&(d.x==c.x)) return true;

if ((b.x==a.x)||(d.x==c.x)) return false;

if ((a.y-b.y)\*(c.x-d.x)==

(c.y-d.y)\*(a.x-b.x)) return true;

return false;

}

bool **nparallel**(line x,line y)

{

return (fabs(x.a\*y.b-y.a\*x.b)<E);

}

pt **intersection**(const pt& a,const pt& b,const pt& c,const pt& d)

{

double A=a.y-b.y,B=b.x-a.x,

C=b.x\*a.y-a.x\*b.y;

double D=c.y-d.y,E=d.x-c.x,

F=d.x\*c.y-c.x\*d.y;

pt e;

e.y=(A\*F-C\*D)/(A\*E-B\*D);

e.x=(C\*E-B\*F)/(A\*E-B\*D);

return e;

}

pt **nintersection**(line x,line y)

{

pt p;

p.x=(y.b\*x.c-x.b\*y.c)/(y.a\*x.b-x.a\*y.b);

p.y=(x.a\*y.c-y.a\*x.c)/(y.a\*x.b-x.a\*y.b);

return p;

}

double **dist2**(const pt& a,const pt& b)

{

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

(b.y-a.y)\*(b.y-a.y));

}

// point line distance, not yet tested

double **pldist**(pt p, line l)

{

double t1 = l.a\*p.x+l.b\*p.y+l.c;

double t2 = sqrt(l.a\*l.a+l.b\*l.b);

return fabs(t1/t2);

}

// it's true when ac is on the left of ab

double **isleft**(const pt& a,const pt& b,const pt& c)

{

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

(b.y-a.y)\*(c.x-a.x);

}

bool **operator<**(

const pt& a,const pt &b)

{

return

(isleft(P[0],b,a)>0||

(isleft(P[0],b,a)==0&&

dist2(P[0],a)<dist2(P[0],b)));

}

vector<pt> P; //need isleft,operator<,dist2,pt

vector<pt>

**convexhull**(const vector<pt>& p)

{

P=p;

vector<pt>S;

double x=P[0].x,y=P[0].y;

int z=0;

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

if (P[i].y<y||

y==P[i].y&&P[i].x<x){

x=P[i].x;

y=P[i].y;

z=i;

}

swap(P[0],P[z]);

sort(P.begin()+1,P.end());

S.push\_back(P[0]);

S.push\_back(P[1]);

for(int i=2;i<P.size();++i){

while (S.size()>=2 &&

isleft(S[S.size()-2],

P[i],S[S.size()-1])<=0)

S.pop\_back();

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

}

return S;

}

// remember to add the first element to the back!!!

double **area**(vector<pt> a)

{

double t=0.0;

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

t+=a[i].x\*a[i+1].y-a[i+1].x\*a[i].y;

return abs(t)/2;

}

struct **circle**{

double g,f,e;

};

circle **makecircle**(pt center,double r){

circle c;

c.g=-1\*center.x;

c.f=-1\*center.y;

c.e=c.f\*c.f+c.g\*c.g-r\*r;

return c;

}

pt **lcintersection**(line l,circle cc)

{

double a,b,c,d;

pt p;

a=l.a\*l.a+l.b\*l.b;

b=2\*(l.a\*l.c-l.a\*l.b\*cc.f+cc.g\*l.b\*l.b);

c=l.c\*l.c-2\*cc.f\*l.b\*l.c+cc.e\*l.b\*l.b;

d=b\*b-4\*a\*c;

if (d<0) return p;

p.x=(sqrt(d)-b)/(2\*a);

if (fabs(l.b)<E){

a=l.a\*l.a+l.b\*l.b;

b=2\*(l.b\*l.c-l.a\*l.b\*cc.g+cc.f\*l.a\*l.a);

c=l.c\*l.c-2\*cc.g\*l.a\*l.c+cc.e\*l.a\*l.a;

d=b\*b-4\*a\*c;

p.y=(sqrt(d)-b)/(2\*a);

}

else

p.y=(p.x\*l.a+l.c)/(-1\*l.b);

return p;

}

Distance Between Two Line Segments

1. If the two lines segments L1(A,B) and L2(C,D) intersect, return 0
2. 2. d min{dist(A,C), dist(B,C), dist(A,D), dist(B,D)}
3. If isAcute(6 ACD) and isAcute(6 ADC) then d min{d, dist(A,L2)}
4. If isAcute(6 BCD) and isAcute(6 BDC) then d min{d, dist(B,L2)}
5. If isAcute(6 CAB) and isAcute(6 CBA) then d min{d, dist(C,L1)}
6. If isAcute(6 DAB) and isAcute(6 DBA) then d min{d, dist(D,L1)}
7. Return d

Acute Angle

• Requires Squared Distance

• Returns a postive number if 6 abc is acute, a negative number if obtuse, and 0 if right-angled

int isAcute(Pt &a, Pt &b, Pt &c) {

return distSqr(a,b) + distSqr(b,c) - distSqr(a,c);

}

Closest Pair Problem

#include<iostream>

#include<algorithm>

#include<vector>

#include<cmath>

#include<iomanip>

using namespace std;

#define maxd 10001

struct pt{

double x,y;

};

typedef vector<pt> vp;

double **dist**(const pt& a, const pt & b){

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

}

bool **xcomp**(const pt &a,const pt &b){

if(a.x==b.x) return (a.y<b.y);

return (a.x<b.x);

}

bool **ycomp**(const pt &a,const pt &b){

if(a.y==b.y) return (a.x<b.x);

return (a.y<b.y);

}

double **closest**(vp &vx, vp &vy){

int n=vx.size();

if(n==1)

{

vy[0]=vx[0];

return maxd;

}

int n1=n/2,n2=n-n/2;

vp vx1(n1), vx2(n2), vy1(n1), vy2(n2);

vp::iterator middle = vx.begin();

advance(middle, n1);

copy(vx.begin(), middle, vx1.begin());

copy(middle, vx.end(), vx2.begin());

double delta = min(closest(vx1,vy1),closest(vx2,vy2));

double mind = delta;

merge(vy1.begin(), vy1.end(), vy2.begin(), vy2.end(), vy.begin(), ycomp);

vp v;

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

if(fabs(vy[i].x-middle->x)<delta)

v.push\_back(vy[i]);

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

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

{

double d=fabs(v[i].y-v[j].y);

if(d>=delta) break;

mind=min(dist(v[i],v[j]),mind);

}

return mind;

}

[Before calling:

vp vx( n ),vy( n );

read points into vx

sort(vx.begin(),vx.end(),xcomp); ]

## Polygon Intersection

// Problem 137: Polygons

//

// One of my early co-geom problem

//

// Find area of "xor" of two polygons

// involves finding intersection of two polygons

//

// algorithm: probably find all intersection points

// then find convex hull. looks stupid

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

#include <iomanip>

using namespace std;

const double E=1e-8;

const double pi=atan(1)\*4;

const double OUTX=1001, OUTY=5009;

typedef struct Coordinate {

double x,y;

};

const Coordinate outside={x:OUTX,y:OUTY};

typedef struct Equation {

double a,b,c;

Equation(const Coordinate& a, const Coordinate& b):

a(a.y-b.y), b(b.x-a.x), c(a.x\*b.y-b.x\*a.y) {};

double evaluate(const Coordinate& p) {

return a\*p.x+b\*p.y+c;

}

};

istream& operator>> (istream& is, Coordinate& a) {

return is >> a.x >> a.y;

}

bool operator== (const Coordinate& a, const Coordinate& b) {

return a.x==b.x && a.y==b.y;

}

bool inpoly(const Coordinate& p, const vector<Coordinate>& a) {

int cnt=0;

for (unsigned i=0; i<a.size()-1; i++)

if (a[i].y<p.y+E && p.y<a[i+1].y-E ||

a[i+1].y<p.y+E && p.y<a[i].y-E) {

double v=(p.y-a[i].y)/(a[i+1].y-a[i].y);

if (p.x<a[i].x+v\*(a[i+1].x-a[i].x)-E)

cnt++;

}

return cnt%2;

}

double ang(const Coordinate& a, const Coordinate& b) {

double t=atan2(b.y-a.y,b.x-a.x);

if (t<0) t+=2\*pi;

return t;

}

double dist(const Coordinate& a, const Coordinate& b) {

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

}

double area(const vector<Coordinate>& a) {

double t=0.0;

for (unsigned i=0; i<a.size()-1; i++)

t+=a[i].x\*a[i+1].y-a[i+1].x\*a[i].y;

return abs(t)/2;

}

int main() {

for (;;) {

int n;

cin >> n;

if (n==0) break;

vector<Coordinate> a(n+1);

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

cin >> a[i];

a[n]=a[0];

cin >> n;

vector<Coordinate> b(n+1);

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

cin >> b[i];

b[n]=b[0];

vector<Coordinate> v;

for (unsigned i=0; i<a.size()-1; i++)

if (inpoly(a[i],b)) {

// cout << '$' << a[i].x << ' ' << a[i].y << endl;

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

}

for (unsigned i=0; i<b.size()-1; i++)

if (inpoly(b[i],a)) {

// cout << '$' << b[i].x << ' ' << b[i].y << endl;

v.push\_back(b[i]);

}

for (unsigned i=0; i<a.size()-1; i++) {

Equation e1(a[i],a[i+1]);

for (unsigned j=0; j<b.size()-1; j++) {

Equation e2(b[j],b[j+1]);

double dx=e1.b\*e2.c-e1.c\*e2.b;

double dy=e1.c\*e2.a-e1.a\*e2.c;

double dd=e1.a\*e2.b-e1.b\*e2.a;

if (abs(dd)<E) continue;

double xx=dx\*1.0/dd;

double yy=dy\*1.0/dd;

if (min(a[i].x,a[i+1].x)-E<xx && xx<max(a[i].x,a[i+1].x)+E &&

min(a[i].y,a[i+1].y)-E<yy && yy<max(a[i].y,a[i+1].y)+E &&

min(b[j].x,b[j+1].x)-E<xx && xx<max(b[j].x,b[j+1].x)+E &&

min(b[j].y,b[j+1].y)-E<yy && yy<max(b[j].y,b[j+1].y)+E) {

v.push\_back();

v.back().x=xx;

v.back().y=yy;

// cout << '#' << xx << ' ' << yy << endl;

}

}

}

// for (unsigned i=0; i<v.size(); i++)

// cout << '$' << v[i].x << ' ' << v[i].y << endl;

// cout << endl;

// find convex hull

vector<Coordinate> w(1);

if (v.size()>0) {

w[0].y=1e38;

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

if (v[i].y<w[0].y-E || (abs(v[i].y-w[0].y)<E && v[i].x<w[0].x-E))

w[0]=v[i];

int m=0;

double angle=-1.0;

do {

double minangle=101;

double d;

int mi;

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

if (v[i]==w[m]) continue;

double a=ang(w[m],v[i]);

double dd=dist(w[m],v[i]);

if (angle<a && (a<minangle-E || abs(a-minangle)<E && d<dd)) {

minangle=a;

d=dd;

mi=i;

}

}

w.push\_back(v[mi]);

m++;

angle=minangle;

} while (!(w[0]==w[m]));

}

//for (unsigned i=0; i<w.size(); i++)

// cout << '$' << w[i].x << ' ' << w[i].y << endl;

// cout << endl;

double a1=area(a);

double a2=area(b);

double a3;

if (v.size()>0)

a3=area(w);

else

a3=0.0;

double ans=a1+a2-2\*a3;

cout << setiosflags(ios::fixed) << setw(8) << setprecision(2) << ans;

// cout << endl;

}

cout << endl;

return 0;

}

## Point inside polygon

// algorithm: count intersection with polygon

// odd means inside, otherwise not

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

const double EE=1e-7;

typedef struct Coordinate {

int x,y;

};

istream& operator>> (istream& is, Coordinate& a) {

return is >> a.x >> a.y;

}

typedef struct Line {

Coordinate a,b;

Line(const Coordinate& a, const Coordinate& b):

a(a),b(b) {};

};

bool intersect(const Line& a, const Line& b) {

int A=a.a.y-a.b.y;

int B=a.b.x-a.a.x;

int C=a.a.x\*a.b.y-a.b.x\*a.a.y;

int D=b.a.y-b.b.y;

int E=b.b.x-b.a.x;

int F=b.a.x\*b.b.y-b.b.x\*b.a.y;

int dd=A\*E-B\*D;

int dx=B\*F-C\*E;

int dy=C\*D-A\*F;

if (dd==0) return false;

double x=dx\*1.0/dd;

double y=dy\*1.0/dd;

return (min(a.a.x,a.b.x)<x+EE && x-EE<max(a.a.x,a.b.x) &&

min(a.a.y,a.b.y)<y+EE && y-EE<max(a.a.y,a.b.y) &&

min(b.a.x,b.b.x)<x+EE && x-EE<max(b.a.x,b.b.x) &&

min(b.a.y,b.b.y)<y+EE && y-EE<max(b.a.y,b.b.y));

}

bool inpolygon(const vector<Coordinate>& x, const Coordinate& p) {

Coordinate q;

q.x=1001; q.y=10001;

Line l1(p,q);

int cnt=0;

for (unsigned i=0; i<x.size()-1; i++) {

Line l2(x[i],x[i+1]);

if (intersect(l1,l2))

cnt++;

}

return cnt&1;

}

int main() {

int n;

cin >> n;

vector<Line> submarine;

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

Coordinate head,tail;

cin >> head >> tail;

submarine.push\_back(Line(head,tail));

}

int m;

cin >> m;

vector<vector<Coordinate> > island(m);

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

int k;

cin >> k;

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

Coordinate p;

cin >> p;

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

}

island[i].push\_back(island[i][0]);

}

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

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

for (unsigned k=0; k<island[j].size()-1; k++) {

Line l(island[j][k],island[j][k+1]);

if (intersect(l,submarine[i]))

goto hell;

}

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

if (inpolygon(island[j],submarine[i].a))

goto heaven;

cout << "Submarine " << i+1 << " is still in water." << endl;

continue;

heaven:

cout << "Submarine " << i+1 << " is completely on land." << endl;

continue;

hell:

cout << "Submarine " << i+1 << " is partially on land." << endl;

}

return 0;

}

## Distance between two polygons

#include <iostream>

#include <cmath>

#include <vector>

#include <algorithm>

#include <iomanip>

using namespace std;

typedef struct Point {

int x,y;

Point(int x, int y): x(x),y(y) {};

};

typedef vector<Point> Polygon;

long double dist(const Point& a, const Point& b) {

// distance between points A and B

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

}

long double dist(const Point& a, const Point& b, const Point& c) {

// distance between line segment AB and point C

const int &x1=a.x, &y1=a.y, &x2=b.x, &y2=b.y, &x=c.x, &y=c.y;

long double A=y1-y2,B=x2-x1,C=x1\*y2-x2\*y1;

long double aa=dist(c,a),bb=dist(c,b),cc=dist(a,b);

if (bb\*bb+cc\*cc-aa\*aa>-1e-10 && aa\*aa+cc\*cc-bb\*bb>-1e-10)

return abs(A\*x+B\*y+C)/sqrt(A\*A+B\*B);

return min(aa,bb);

}

long double dist(const Polygon& a, const Polygon& b) {

long double ans=1e10;

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

for (unsigned j=0; j<b.size(); j++) {

long double d1=dist(a[i],a[(i+1)%a.size()],b[j]);

long double d2=dist(a[i],a[(i+1)%a.size()],b[(j+1)%b.size()]);

long double d3=dist(b[j],b[(j+1)%b.size()],a[i]);

long double d4=dist(b[j],b[(j+1)%b.size()],a[(i+1)%a.size()]);

if (ans>d1)

ans=d1;

if (ans>d2)

ans=d2;

if (ans>d3)

ans=d3;

if (ans>d4)

ans=d4;

}

return ans;

}

int main() {

int T;

cin >> T;

while (T--) {

int r1,r2,n;

cin >> r1 >> r2 >> n;

vector<Polygon> island;

Polygon p,q;

for (int i=0; i<r1; i++) {

int x,y;

cin >> x >> y;

p.push\_back(Point(x,y));

}

for (int i=0; i<r2; i++) {

int x,y;

cin >> x >> y;

q.push\_back(Point(x,y));

}

if (p[0].y<q[0].y) {

island.push\_back(p);

island.push\_back(q);

} else {

island.push\_back(q);

island.push\_back(p);

}

if (true) {

int x1=island[0][0].x, x2=island[0].back().x;

island[0].push\_back(Point(x2,-1));

island[0].push\_back(Point(x1,-1));

}

if (true) {

int x1=island[1][0].x, x2=island[1].back().x;

island[1].push\_back(Point(x2,10001));

island[1].push\_back(Point(x1,10001));

}

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

int m;

cin >> m;

island.push\_back();

while (m--) {

int x,y;

cin >> x >> y;

island.back().push\_back(Point(x,y));

}

}

if (true) {

long double x[20][20];

for (unsigned i=0; i<island.size(); i++) {

x[i][i]=0;

for (unsigned j=i+1; j<island.size(); j++)

x[i][j]=x[j][i]=dist(island[i],island[j]);

}

// for (unsigned i=0; i<island.size(); i++) {

// for (unsigned j=0; j<island.size(); j++)

// cout << x[i][j] << ' ';

// cout << endl;

// }

int z[20][20];

memset(z,255,sizeof(z));

for (unsigned k=0; k<island.size(); k++)

for (unsigned i=0; i<island.size(); i++)

for (unsigned j=0; j<island.size(); j++) {

if (x[i][j]>x[i][k]+x[k][j]) {

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

z[i][j]=k;

}

}

// cout << "-----" << endl;

// for (unsigned i=0; i<island.size(); i++) {

// for (unsigned j=0; j<island.size(); j++)

// cout << x[i][j] << ' ';

// cout << endl;

// }

// cout << "-----" << endl;

// for (unsigned i=0; i<island.size(); i++) {

// for (unsigned j=0; j<island.size(); j++)

// cout << z[i][j] << ' ';

// cout << endl;

// }

cout << setiosflags(ios::fixed) << setprecision(3) << x[0][1] << endl;

}

}

return 0;

}

## Distance between line segments

#include <iostream>

#include <cmath>

#include <vector>

#include <algorithm>

#include <iomanip>

#include <string>

using namespace std;

const long double E=1e-8;

struct Point {

long double x,y;

Point(long double x=0, long double y=0): x(x),y(y) {};

};

istream& operator>> (istream& is, Point& a) {

return is >> a.x >> a.y;

}

long double dist(const Point& a, const Point& b) {

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

}

long double dist(const Point& a, const Point& b, const Point& c) {

const long double &x1=a.x, &y1=a.y, &x2=b.x, &y2=b.y, &x=c.x, &y=c.y;

long double A=y1-y2,B=x2-x1,C=x1\*y2-x2\*y1;

long double aa=dist(c,a),bb=dist(c,b),cc=dist(a,b);

if (bb\*bb+cc\*cc-aa\*aa>-1e-10 && aa\*aa+cc\*cc-bb\*bb>-1e-10)

return fabs(A\*x+B\*y+C)/sqrt(A\*A+B\*B);

return min(aa,bb);

}

struct Line {

long double A,B,C;

Line (const Point& a, const Point& b) {

A=a.y-b.y;

B=b.x-a.x;

C=a.x\*b.y-b.x\*a.y;

}

long double dist(const Point& p) {

return fabs((A\*p.x+B\*p.y+C)/sqrt(A\*A+B\*B));

}

};

Point operator\* (const Line& a, const Line& b) {

long double dx,dy,dd;

dd=a.A\*b.B-b.A\*a.B;

dx=a.B\*b.C-b.B\*a.C;

dy=a.C\*b.A-b.C\*a.A;

if (fabs(dd)<E) {

if (fabs(dx)<E || fabs(dy)<E)

throw 1;

throw 0;

}

return Point(dx/dd,dy/dd);

}

inline bool between(long double a, long double b, long double c) {

// c between a and b?

return a-E<c && c<b+E || a+E>c && c>b-E;

}

inline bool between(const Point& a, const Point& b, const Point& c) {

// c between a and b?

return between(a.x,b.x,c.x) && between(a.y,b.y,c.y);

}

int main() {

Point p1,p2,q1,q2;

string p,q;

for (;;) {

cin >> p1 >> p2 >> p >> q1 >> q2 >> q;

if (p=="END" && q=="END") break;

if (p=="LS" && q=="LS") {

Line l1(p1,p2);

Line l2(q1,q2);

try {

Point t=l1\*l2;

if (between(p1,p2,t) && between(q1,q2,t))

cout << "0.00000" << endl;

else

throw 0;

} catch (int k) {

if (k==0) {

long double d1=dist(p1,p2,q1);

long double d2=dist(p1,p2,q2);

long double d3=dist(q1,q2,p1);

long double d4=dist(q1,q2,p2);

long double ans=min(min(d1,d2),min(d3,d4));

cout << setiosflags(ios::fixed)

<< setprecision(5) << ans << endl;

} else {

if (between(p1,p2,q1) || between(p1,p2,q2) ||

between(q1,q2,p1) || between(q1,q2,p2))

cout << "0.00000" << endl;

else {

long double d1=dist(p1,p2,q1);

long double d2=dist(p1,p2,q2);

long double d3=dist(q1,q2,p1);

long double d4=dist(q1,q2,p2);

long double ans=min(min(d1,d2),min(d3,d4));

cout << setiosflags(ios::fixed)

<< setprecision(5) << ans << endl;

}

}

}

} else if (p=="L" && q=="L") {

Line l1(p1,p2);

Line l2(q1,q2);

try {

Point t=l1\*l2;

cout << "0.00000" << endl;

} catch (int k) {

if (k==0) {

long double c1=l1.C/sqrt(l1.A\*l1.A+l1.B\*l1.B);

long double c2=l2.C/sqrt(l2.A\*l2.A+l2.B\*l2.B);

cout << setiosflags(ios::fixed)

<< setprecision(5) << fabs(c1-c2) << endl;

} else {

cout << "0.00000" << endl;

}

}

} else {

if (q=="LS") {

swap(p1,q1);

swap(p2,q2);

swap(p,q);

}

Line l1(p1,p2);

Line l2(q1,q2);

try {

Point t=l1\*l2;

if (between(p1,p2,t))

cout << "0.00000" << endl;

else {

long double d1=l2.dist(p1);

long double d2=l2.dist(p2);

long double ans=min(d1,d2);

cout << setiosflags(ios::fixed)

<< setprecision(5) << ans << endl;

}

} catch (int k) {

if (k==0) {

long double c1=l1.C/sqrt(l1.A\*l1.A+l1.B\*l1.B);

long double c2=l2.C/sqrt(l2.A\*l2.A+l2.B\*l2.B);

cout << setiosflags(ios::fixed)

<< setprecision(5) << fabs(c1-c2) << endl;

} else {

cout << "0.00000" << endl;

}

}

}

}

return 0;

}

## Polygon Symmetry

#include <iostream>

using namespace std;

long long x[2000],y[2000];

struct Line {

long long a,b,c;

Line(long long x1,long long y1,long long x2,long long y2) {

a=y1-y2;

b=x2-x1;

c=x1\*y2-x2\*y1;

}

long long evaluate(long long x, long long y) {

return a\*x+b\*y+c;

}

};

bool perpendicular(const Line& l1, const Line& l2) {

return l1.a\*l2.a==-l1.b\*l2.b;

}

int main() {

int cnt=0;

for (;;) {

int n;

cin >> n;

if (n==0) break;

if (n>1000) throw 0;

for (int i=0; i<2\*n; i+=2) {

cin >> x[i] >> y[i];

x[i]\*=2;

y[i]\*=2;

}

for (int i=0; i<2\*n; i+=2) {

x[i+1]=(x[i]+x[(i+2)%(2\*n)])/2;

y[i+1]=(y[i]+y[(i+2)%(2\*n)])/2;

}

int ans=0;

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

Line l(x[i],y[i],x[i+n],y[i+n]);

int a=(i+2\*n-1)%(2\*n);

int b=(i+1)%(2\*n);

while (a!=b) {

long long d1=l.evaluate(x[a],y[a]);

long long d2=l.evaluate(x[b],y[b]);

if (d1!=-d2)

goto hell;

Line p(x[a],y[a],x[b],y[b]);

if (!perpendicular(l,p))

goto hell;

a=(a+2\*n-1)%(2\*n);

b=(b+1)%(2\*n);

}

ans++;

hell:

continue;

}

cout << "Polygon #" << ++cnt << " has " << ans << " symmetry line(s)." << endl;

}

return 0;

}

## Rotation and Resize

#include <iostream>

#include <cmath>

#include <set>

#include <vector>

#include <string>

#include <iomanip>

#include <algorithm>

using namespace std;

typedef struct rec {

int x,y,z;

};

istream& operator>> (istream& is, rec& a) {

return is>>a.x>>a.y>>a.z;

}

ostream& operator<< (ostream& os, const rec& a) {

return os <<'('<< setiosflags(ios::fixed)

<< setprecision(0) << a.x << ','

<< setprecision(0) << a.y <<')';

}

bool operator< (const rec& a, const rec& b) {

return a.x<b.x || a.x==b.x && a.y<b.y;

}

bool operator== (const rec& a, const rec& b) {

return a.x==b.x && a.y==b.y;

}

typedef struct Coordinate {

int x,y;

};

istream& operator>> (istream& is, Coordinate& a) {

return is>>a.x>>a.y;

}

inline int sqr(int x) {

return x\*x;

};

void printarray(const vector<rec>& x) {

for (unsigned i=0; i<x.size(); i++)

cout << x[i].x << ',' << x[i].y << ' ';

cout << endl;

}

int main() {

int cnt=0;

for (;;) {

int n;

cin >> n; cin.ignore(2147483647,'\n');

if (n==0) break;

vector<rec> x(n);

set<rec> y;

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

cin >> x[i]; cin.ignore(2147483647,'\n');

y.insert(x[i]);

}

int m;

cin >> m; cin.ignore(2147483647,'\n');

vector<vector<Coordinate> > z;

vector<string> name;

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

int size; string s;

cin >> size >> s; cin.ignore(2147483647,'\n');

z.push\_back(vector<Coordinate>(size));

name.push\_back(s);

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

cin >> z[i][j]; cin.ignore(2147483647,'\n');

}

for (int j=size-1; j>=0; j--) {

z[i][j].x-=z[i][0].x;

z[i][j].y-=z[i][0].y;

}

}

cout << "Map #" << ++cnt << endl;

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

set<vector<rec> > star;

if (z[k].size()==1) {

// match anyway :P

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

star.insert(vector<rec>(1,x[i]));

} else {

// need searching this time

double olen=sqrt(sqr(z[k][1].x)+sqr(z[k][1].y));

double otheta=atan2(z[k][1].y,z[k][1].x);

vector<rec> mystar(z[k].size());

vector<rec> newstar;

set<rec>::iterator itr;

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

mystar[0]=x[i];

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

if (i==j) continue;

mystar[1]=x[j];

double nlen=sqrt(sqr(x[j].x-x[i].x)+sqr(x[j].y-x[i].y));

double ntheta=atan2(x[j].y-x[i].y,x[j].x-x[i].x);

double theta=ntheta-otheta;

double scale=nlen/olen;

for (unsigned l=2; l<z[k].size(); l++) {

double newx,newy;

newx=z[k][l].x\*cos(theta)-z[k][l].y\*sin(theta);

newy=z[k][l].x\*sin(theta)+z[k][l].y\*cos(theta);

newx\*=scale; newy\*=scale;

newx+=x[i].x; newy+=x[i].y;

if (abs(newx-int(floor(newx+0.49999999)))>1e-7 ||

abs(newy-int(floor(newy+0.49999999)))>1e-7)

goto skip;

rec r;

r.x=int(floor(newx+0.49999999)); r.y=int(floor(newy+0.49999999));

if ((itr=y.find(r))==y.end()) goto skip;

mystar[l]=\*itr;

}

// printarray(mystar);

newstar=mystar;

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

star.insert(newstar);

skip:

continue;

}

}

/\*

// cout << "uhoh" << endl;

for (unsigned l=0; l<z[k].size(); l++)

z[k][l].y=-z[k][l].y;

otheta=atan2(z[k][1].y,z[k][1].x);

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

mystar[0]=x[i];

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

if (i==j) continue;

mystar[1]=x[j];

double nlen=sqrt(sqr(x[j].x-x[i].x)+sqr(x[j].y-x[i].y));

double ntheta=atan2(x[j].y-x[i].y,x[j].x-x[i].x);

double theta=ntheta-otheta;

double scale=nlen/olen;

for (unsigned l=2; l<z[k].size(); l++) {

double newx,newy;

newx=z[k][l].x\*cos(theta)-z[k][l].y\*sin(theta);

newy=z[k][l].x\*sin(theta)+z[k][l].y\*cos(theta);

newx\*=scale; newy\*=scale;

newx+=x[i].x; newy+=x[i].y;

if (abs(newx-int(floor(newx+0.49999999)))>1e-7 ||

abs(newy-int(floor(newy+0.49999999)))>1e-7)

goto skip2;

rec r;

r.x=int(floor(newx+0.49999999)); r.y=int(floor(newy+0.49999999));

if ((itr=y.find(r))==y.end()) goto skip2;

mystar[l]=\*itr;

}

// printarray(mystar);

newstar=mystar;

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

star.insert(newstar);

skip2:

continue;

}

}

\*/

}

cout << endl;

cout << name[k] << " occurs " << star.size()

<< " time(s) in the map." << endl;

if (!star.empty()) {

set<vector<rec> >::iterator itr,best;

int bestvalue=-1;

for (itr=star.begin(); itr!=star.end(); itr++) {

int sum=0;

for (unsigned i=0; i<itr->size(); i++) sum+=(\*itr)[i].z;

// printarray(\*itr);

if (sum>bestvalue) {

bestvalue=sum;

best=itr;

}

}

cout << "Brightest occurrence:";

for (unsigned i=0; i<best->size(); i++)

cout << ' ' << (\*best)[i];

cout << endl;

}

}

cout << "-----" << endl;

}

return 0;

}

## Polygon, eating d cm of its edges

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

inline double dist(double x1, double y1, double x2, double y2) {

return sqrt((x2-x1)\*(x2-x1)+(y2-y1)\*(y2-y1));

}

int main() {

for (;;) {

double d;

int n;

cin >> d >> n;

if (n==0) break;

double x[10000],y[10000];

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

cin >> x[i] >> y[i];

double a1=0.0;

double a2=0.0;

double a3=0.0;

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

a1+=x[i]\*y[(i+1)%n]-x[(i+1)%n]\*y[i];

double a,b,c,theta,sigma;

a=dist(x[i],y[i],x[(i+1)%n],y[(i+1)%n]);

b=dist(x[i],y[i],x[(i+n-1)%n],y[(i+n-1)%n]);

c=dist(x[(i+1)%n],y[(i+1)%n],x[(i+n-1)%n],y[(i+n-1)%n]);

theta=acos((a\*a+b\*b-c\*c)/2/a/b);

a=dist(x[i],y[i],x[(i+1)%n],y[(i+1)%n]);

b=dist(x[(i+1)%n],y[(i+1)%n],x[(i+2)%n],y[(i+2)%n]);

c=dist(x[i],y[i],x[(i+2)%n],y[(i+2)%n]);

sigma=acos((a\*a+b\*b-c\*c)/2/a/b);

double q=d/tan(theta/2),r=d/tan(sigma/2);

double p=a-q-r;

a2+=p\*d;

a3+=d\*d/tan(theta/2);

}

a1=abs(a1)/2;

double ans=a1-a2-a3;

// cout << a1 << ' ' << a2 << ' ' << a3 << endl;

cout << setiosflags(ios::fixed)

<< setprecision(3) << ans << endl;

}

return 0;

}

## Smallest Enclosing Disk

MinDisk(P):

If |P| <= 3, then return the disk passing through these points. Otherwise, randomly permute the points in P yielding the sequence <p\_1, p\_2, ..., p\_n>

Let D\_2 be the minimum disk enclosing {p\_1,p\_2}.

for i = 3 to |P| do

if p\_i in D\_{i-1} then D\_i = D\_{i-1}

else D\_i = MinDiskWith1Pt(P[1..i-1],p\_i)

MinDiskWith1Pt(P,q):

Randomly permute the points in P. Let D\_i be the minimum disk enclosing {q,p\_1}.

for i = 2 to |P| do

if p\_i in D\_{i-1} then D\_i = D\_{i-1}

else D\_i = MinDiskWith2Pts(P[1..i-1],q,p\_i)

MinDiskWith2Pts(P,q\_1,q\_2):

Randomly permute the points in P. Let D\_0 be the minimum disk enclosing {q\_1,q\_2}

for i = 1 to |P| do

if p\_i in D\_{i-1} then D­\_i = D\_{i-1}

else D\_i = Disk(q\_1,q\_2,p\_i)

## Complex Numbers

#include <complex>

complex(re, im)

double arg(z) //(-π,π]

complex conj(z)

double imag(z)

double real(z)

double abs(z) // magnitude

double norm(z) //square of magnitude

complex polar(r, theta)

complex exp(z)

complex log(z)

complex pow(a, z) // double a

complex sqrt(z) // sqrt of z in 1st or 4th quadrant

**I/O format:** (1.01, 2.70)

### Complex Number: Points and Lines

z = p + ts (where |s| = 1)

Slope:

imag(s) / real (s)

CCW nomral at p:

z’ = p + t(is)

Point-line distance:

imag((q-p)/s)

Point-point distance:

abs(q-p)

CCW rotation of a point q about origin:

q \* complex(cos(t), sin(t))

Projection of u onto v:

real(u/v)\*v

Inner product:

real(conj(p)\*q)

Cross product:

imag(conj(p)\*q)

Normal form to vector form:

s = complex(b,-a);

p = abs(a) > abs(b) ? complex(-c/a, 0) : complex(0, -c/b);

Vector form to normal form:

a = -imag(s), b =real(s),

c = real(s/p)

Line intersection (z = p + tr and z = q +ts)

xp = cross(r,s); ans = p+r\*cross(q-p,s)/xp

Tangent from p outside the circle:

k = asin(r / abs(p))  
 u = p\*complex(cos(k), sin(k))  
 v = p\*complex(cos(k), -sin(k))  
 u /= abs(u)  
 v /= abs(v)

Circles at Origin: **|z| = r**

Tangent at p (on circumference): **z = p + t(ip)**

### A working dijkstra

#include <iostream>

#include <queue>

#include <vector>

#include <string>

#include <sstream>

using namespace std;

int d[104];

struct dist{

bool operator()(int x,int y){

return d[x]>d[y];

}

};

long abb(long x){

return (x>0)?(x):(-x);

}

priority\_queue<int,vector<int>, dist> pq;

bool a[10][104];

int main(){

int n,k;

int t[10];

while (cin >> n >> k){

memset(a,0,sizeof a);

while (!pq.empty()) pq.pop();

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

cin >> t[i];

string s;

getline(cin,s);

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

string s;

getline(cin,s);

istringstream iss(s);

int x;

while (iss>> x){

a[i][x] = true;

}

}

if (k==0)

{

cout <<0<<endl;

continue;

}

for (int i = 0; i <= 100; ++i)

d[i] = 10000000;

d[0] =-60;

bool v[105];

memset(v,0,sizeof v);

pq.push(0);

while (!pq.empty()){

while (!pq.empty()&&v[pq.top()])

pq.pop();

if (pq.empty()) break;

v[pq.top()] = true;

// cout << pq.top()<<" "<<d[pq.top()]<<endl;

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

if (a[i][pq.top()]){

for (int j = 0; j <=100; ++j){

long xxx = (d[pq.top()]+t[i]\*(abb(pq.top()-j))+60);

// cout << i<<" "<< j<<" "<<d[pq.top()]<<" "<<d[j]<<" "<<xxx<<" "<<(d[j]<xxx)<<endl;

if (a[i][j] && !v[j] && d[j]>xxx){

// cout << pq.top()<<" "<<d[pq.top()]<<" "<<j <<" "<< d[j]<<" "<<i<<endl;

d[j] = xxx;

pq.push(j);

}

}

}

pq.pop();

}

if (d[k]<10000000)

cout << d[k]<<endl;

else

cout <<"IMPOSSIBLE\n";

}

}

### Prime Factorisation

int phi(int x){

long m,p;

m = p = x;

for (int i = 2; i\*i<=p; ++i){

if (p%i==0){

// cout <<i<<endl;

m = (m/i\*(i-1));

while (p%i==0) p/=i;

}

}

if (p!=1){

// cout <<p<<endl;

m = (m/p\*(p-1));

}

return m;

}

### binary indexed tree

int a[1000][1000];

void add(int x,int y,int z){

x+=107;

y+=107;

for (int i = x; i <= 300; i+=(i&(-i)))

for (int j = y; j <= 300; j+=(j& (-j)))

a[i][j]+=z;

}

int get(int x,int y){

x+=107;

y+=107;

int ans = 0;

for (int i = x; i>=1; i-=(i&(-i)))

for (int j = y; j >=1; j-=(j&(-j)))

ans+=a[i][j];

return ans;

}

Binary search tree

Two versions of top sort

### SCC

Input: Graph G = (V, E), Start node v0

index = 0 // DFS node number counter

S = empty // An empty stack of nodes

tarjan(v0) // Start a DFS at the start node

procedure tarjan(v)

v.index = index // Set the depth index for v

v.lowlink = index

index = index + 1

S.push(v) // Push v on the stack

forall (v, v') in E do{

if (v'.index is undefined){

tarjan(v')

v.lowlink = min(v.lowlink, v'.lowlink)

}

else if (v' in S)

v.lowlink = min(v.lowlink, v'.index)

}

if (v.lowlink == v.index){

print "SCC:"

repeat

v' = S.pop

print v'

until (v' == v)

}

//c++ implementation

#include <iostream>

#include <cstring>

#include <algorithm>

#include <map>

#include <vector>

#include <stack>

#include <sstream>

#include <queue>

using namespace std;

vector<int> a[140];

int vindex[140],vlowlink[140];

int ind = 0;

bool instack[140];

string output[140];

int t= 0;

stack<int> S;

struct minheap{

bool operator()(int a,int b){

return a>b;

}

};

void tarjan(int v){

vindex[v] = ind;

vlowlink[v] = ind;

++ind;

S.push(v);

instack[v] = true;

for (unsigned j = 0; j <a[v].size(); ++j)

if (vindex[(int)a[v][j]]==-1){

tarjan(a[v][j]);

vlowlink[v] = min(vlowlink[v],vlowlink[a[v][j]]);

}

else if (instack[(int)a[v][j]])

vlowlink[v] = min(vlowlink[v],vindex[a[v][j]]);

int vp;

if (vlowlink[v]==vindex[v])

{

bool first = true;

priority\_queue<int,vector<int>,minheap> pq;

ostringstream oss;

do{

vp = S.top();

S.pop();

instack[vp] = false;

pq.push(vp);

} while (vp!=v);

while (!pq.empty()){

if (first){

oss << (char)pq.top();

first = false;

}

else oss <<" "<<(char)pq.top();

pq.pop();

}

oss << endl;

output[t++] = oss.str();

}

}

int main(){

int n;

bool ffff= false;

while (cin >> n,n){

if (ffff) cout <<endl;else ffff = true;

char u;

memset(vindex,-1,sizeof vindex);

memset(vlowlink,-1,sizeof vlowlink);

memset(instack,0,sizeof instack);

for (int i = 'A'; i <='Z'; ++i)

a[i].clear();

while (!S.empty())

S.pop();

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

char p,q,r,s,t;

cin >> p >> q >> r >> s >> t >> u;

// if (u!=p)

a[(int)u].push\_back(p);

// if (u!=q)

a[(int)u].push\_back(q);

// if (u!=r)

a[(int)u].push\_back(r);

// if (u!=s);

a[(int)u].push\_back(s);

// if (u!=t)

a[(int)u].push\_back(t);

}

t = 0;

ind = 0;

for (int i = 'A'; i<='Z'; ++i)

if (vindex[i]==-1 && a[i].size())

tarjan(i);

sort(output,output+t);

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

cout << output[i];

}

}

### Binary Tree (IOI2005-like)

#include <iostream>

#include <algorithm>

#include <utility>

using namespace std;

#define MAXN 100000

typedef pair<int,int> node;

struct treetype{

long long sum, left, right, middle;

} a[4\*(MAXN+1)];//storing tree values

node b[4\*(MAXN+1)];//storing intervals

int c[MAXN+1];

void init(int i,int l,int h){

b[i].first = l;

b[i].second = h;

a[i].sum = 0;

a[i].left = 0;

a[i].right = 0;

a[i].middle = 0;

// cout << i <<" "<<l <<" "<<h<<endl;

if (l<h){

init(i\*2+1,l,(l+h)/2);

init(i\*2+2,(l+h)/2+1,h);

}

}

void add(long long k,int i,int pos){

// cout << k <<" "<<i <<" "<<b[i].first<<" " <<b[i].second<<endl;

a[i].sum+=k;

if (b[i].first==b[i].second){

a[i].left = a[i].sum;

a[i].right = a[i].sum;

a[i].middle = a[i].sum;

return;

}

if (pos<=(b[i].first+b[i].second)/2){

add(k,i\*2+1,pos);

}else{

add(k,i\*2+2,pos);

}

a[i].left = max(a[i\*2+1].left,a[i\*2+1].sum+a[i\*2+2].left);

a[i].right = max(a[i\*2+2].right,a[i\*2+1].right+a[i\*2+2].sum);

a[i].middle = max(max(a[i\*2+1].middle,a[i\*2+2].middle),(a[i\*2+1].right+a[i\*2+2].left));

// cout << "max between "<<b[i].first <<" and "<<b[i].second<<" = "<<a[i].left <<" "<<a[i].right<<" "<<a[i].middle<<endl;

}

int main(){

int n,q;

cin >> n >> q;

init(0,1,n);

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

cin >> c[i];

add(c[i],0,i);

}

cout << a[0].middle <<endl;

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

int p;

long long k;

cin >> p >> k;

long long diff = k-c[p];

add(diff,0,p);

c[p] = k;

cout << a[0].middle<<endl;

}

}

### LCA

#include <iostream>

#include <cmath>

#include <string>

#include <map>

#include <vector>

using namespace std;

map<string,int> m;

int k,n;

vector<int> a[500];

int p[500];

int mk[500];

int d[500];

int orz;

int lca[500][500];

void dfs(int n,int dep){

// cout << n << " " << dep <<endl;

d[n] = dep;

mk[n] = orz;

for (unsigned i = 0; i < a[n].size(); ++i)

if (p[a[n][i]]==n)

dfs(a[n][i],dep+1);

}

int callca(int x,int y){

if (lca[x][y]!=-1) return lca[x][y];

if (x==y) lca[x][y] = x;

else if (d[x]>d[y]) lca[x][y] = callca(p[x],y);

else lca[x][y] = callca(x,p[y]);

return lca[x][y];

}

int main(){

string s,t;

n = 0;

m.clear();

memset(p,-1,sizeof p);

while (cin >> s >> t,s!="no.child"){

if (m.find(s)==m.end()){

m[s] = n;

++n;

}

if (m.find(t)==m.end()){

m[t] = n;

++n;

}

p[m[s]] = m[t];

a[m[t]].push\_back(m[s]);

a[m[s]].push\_back(m[t]);

}

memset(d,-1,sizeof d);

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

if (p[i]==-1)

{

orz = i;

dfs(i,0);

}

memset(lca,-1,sizeof lca);

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

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

if (mk[i]==mk[j]){

callca(i,j);

}

while (cin >> s >> t){

if (m.find(s)==m.end()||m.find(t)==m.end()){

cout << "no relation\n";

continue;

}

if (mk[m[s]]!=mk[m[t]]){

cout << "no relation\n";

continue;

}

if (d[m[s]]>d[m[t]]){

if (lca[m[s]][m[t]]==m[t]){

if (p[m[s]]==m[t]){

cout << "child\n";

continue;

}

int dd = d[m[s]]-d[m[t]];

for (int i = dd; i>2; --i)

cout << "great ";

cout << "grand child\n";

continue;

}

else {

int lcc = lca[m[s]][m[t]];

int ans1 = min(d[m[s]]-1-d[lcc],d[m[t]]-1-d[lcc]);

int ans2 = abs(d[m[s]]-d[m[t]]);

cout << ans1 <<" cousin removed " << ans2 <<endl;

// cout << lcc <<" "<<d[lcc]<<" " <<d[m[s]]<<" " << d[m[t]]<<endl;

continue;

}

} else{

if (p[m[s]]==p[m[t]]){

cout << "sibling\n";

continue;

} else{

if (d[m[s]]==d[m[t]]||lca[m[s]][m[t]]!=m[s]){

int lcc = lca[m[s]][m[t]];

int ans1 = min(d[m[s]]-1-d[lcc],d[m[t]]-1-d[lcc]);

int ans2 = abs(d[m[s]]-d[m[t]]);

if (ans2!=0){

// cout << lcc <<" "<<d[lcc]<<" " <<d[m[s]]<<" " << d[m[t]]<<endl;

cout << ans1 <<" cousin removed " << ans2 <<endl;

} else{

// cout << lcc <<" "<<d[lcc]<<" " <<d[m[s]]<<" " << d[m[t]]<<endl;

cout << ans1 <<" cousin" <<endl;

}

continue;

}

if (lca[m[s]][m[t]]==m[s]){

if (p[m[t]]==m[s]){

cout << "parent\n";

continue;

}

int dd = d[m[t]]-d[m[s]];

for (int i = dd; i>2; --i)

cout << "great ";

cout << "grand parent\n";

continue;

}

}

}

}

}