This document contains assorted algorithmic implementations for various well known problems from ACM ICPC style programming contests and it consists of 25 pages including the cover page.

**Team Members**

Momontho Mashak Monmoy

Muhammad Ridowan

Zobayer Hasan

Du fruit ninjas

Team Notebook

**Graph:**

1. **Maximum Flow (Dinitz)**

// works on undirected graph can have loops, multiple edges, cycles

int src, snk, nNode, nEdge; int Q[MAXN], fin[MAXN], pro[MAXN], dist[MAXN]; int flow[MAXE], cap[MAXE], next[MAXE], to[MAXE];

inline void init(int \_src, int \_snk, int \_n) {

    src = \_src, snk = \_snk, nNode = \_n, nEdge = 0; SET(fin);

}

inline void add(int u, int v, int \_cap) {

    to[nEdge] = v, cap[nEdge] = \_cap, flow[nEdge] = 0; next[nEdge] = fin[u], fin[u] = nEdge++; to[nEdge] = u, cap[nEdge] = \_cap, flow[nEdge] = 0;

    next[nEdge] = fin[v], fin[v] = nEdge++;

}

bool bfs() {

    int st, en, i, u, v; SET(dist); dist[src] = st = en = 0; Q[en++] = src;

    while(st < en) {

        u = Q[st++];

        for(i=fin[u]; i>=0; i=next[i]) {

            v = to[i];

            if(flow[i] < cap[i] && dist[v]==-1) {

                dist[v] = dist[u]+1; Q[en++] = v;

            } } }

    return dist[snk]!=-1; }

int dfs(int u, int fl) {

    if(u==snk) return fl;

    for(int &e=pro[u], v, df; e>=0; e=next[e]) { v = to[e];

        if(flow[e] < cap[e] && dist[v]==dist[u]+1) {

            df = dfs(v, min(cap[e]-flow[e], fl));

            if(df>0) {

                flow[e] += df; flow[e^1] -= df; return df;

            } } }

    return 0;

}

i64 dinitz() {

    i64 ret = 0;

    int df;

    while(bfs()) {

        for(int i=1; i<=nNode; i++) pro[i] = fin[i];

        while(true) {

            df = dfs(src, INF); if(df) ret += (i64)df; else break;

        } }

    return ret;

}

1. **Min Cost Max Flow (Bellman Ford)**

// works only on directed graphs handles multiple edges, cycles, loops

int src, snk, nNode, nEdge; int fin[MAXN], pre[MAXN], dist[MAXN]; int cap[MAXE], cost[MAXE], next[MAXE], to[MAXE], from[MAXE];

inline void init(int \_src, int \_snk, int nodes) {

    SET(fin); nNode = nodes, nEdge = 0; src = \_src, snk = \_snk;

}

inline void addEdge(int u, int v, int \_cap, int \_cost) {

    from[nEdge] = u, to[nEdge] = v, cap[nEdge] = \_cap, cost[nEdge] = \_cost; next[nEdge] = fin[u], fin[u] = nEdge++;

    from[nEdge] = v, to[nEdge] = u, cap[nEdge] = 0, cost[nEdge] = -(\_cost); next[nEdge] = fin[v], fin[v] = nEdge++;

}

bool bellman() {

    int iter, u, v, i; bool flag = true;

    MEM(dist, 0x7f); SET(pre);

    dist[src] = 0;

    for(iter = 1; iter < nNode && flag; iter++) {

        flag = false;

        for(u = 0; u < nNode; u++) {

            for(i = fin[u]; i >= 0; i = next[i]) {

                v = to[i];

                if(cap[i] && dist[v] > dist[u] + cost[i]) {

                    dist[v] = dist[u] + cost[i]; pre[v] = i; flag = true;

                }  } } }

    return (dist[snk] < INF);

}

int mcmf(int &fcost) {

    int netflow, i, bot, u; netflow = fcost = 0;

    while(bellman()) {

        bot = INF;

        for(u = pre[snk]; u >= 0; u = pre[from[u]]) bot = min(bot, cap[u]);

        for(u = pre[snk]; u >= 0; u = pre[from[u]]) {

            cap[u] -= bot; cap[u^1] += bot; fcost += bot \* cost[u];

        }

        netflow += bot;

    }

    return netflow;

}

1. **Maximum Bipartite Matching (Hopcroft Karp)**

/\* n: number of nodes on left side, nodes are numbered 1 to n, m: number of nodes on right side, nodes are numbered n+1 to n+m, G = NIL[0] ? G1[G[1---n]] ? G2[G[n+1---n+m]] \*/

bool bfs() {

    int i, u, v, len; queue< int > Q;

    for(i=1; i<=n; i++) {

        if(match[i]==NIL) {

            dist[i] = 0; Q.push(i);

        }

        else dist[i] = INF;

    }

    dist[NIL] = INF;

    while(!Q.empty()) {

        u = Q.front(); Q.pop();

        if(u!=NIL) {

            len = G[u].size(); for(i=0; i<len; i++) {

                v = G[u][i];

                if(dist[match[v]]==INF) {

dist[match[v]] = dist[u] + 1; Q.push(match[v]); } }  } }

    return (dist[NIL]!=INF);

}

bool dfs(int u) {

    int i, v, len;

    if(u!=NIL) {

        len = G[u].size();

        for(i=0; i<len; i++) {

            v = G[u][i];

            if(dist[match[v]]==dist[u]+1) {

                if(dfs(match[v])) {

                    match[v] = u; match[u] = v; return true;

                } } }

 dist[u] = INF; return false;

    }

    return true;

}

int hopcroft\_karp() {

    int matching = 0, i;

    CLR(match);

    while(bfs())

        for(i=1; i<=n; i++)

            if(match[i]==NIL && dfs(i)) matching++;

    return matching; }

1. **Heavy-Light Decomposition :**

/\* Also includes the LCA method.\*/

vector< int > G[MAX];

int cost[MAX], lvl[MAX], parent[MAX];

int head[MAX], cnext[MAX], chainid[MAX], chainpos[MAX];

int nchain, temp[MAX];

int dfs(int u, int p, int d) {

    int i, v, sz = G[u].size(), tmp, mx, id, tot, hd, k;

    lvl[u] = d, mx = 0, id = u, tot = 1;

    for(i = 0; i < sz; i++) {

        v = G[u][i]; if(v != p) {

            parent[v] = u;  tmp = dfs(v, u, d + 1);

            tot += tmp; if(tmp > mx) {

                mx = tmp; id = v;

            } } }

    if(tot == 1) cnext[u] = -1;

    else cnext[u] = id;

    for(i = 0; i < sz; i++) {

        v = G[u][i];

        if(v != p && v != id) {

for(hd = v, k = 0; v != -1; v = cnext[v], k++) {

                head[v] = hd; temp[k] = cost[v];

                chainpos[v] = k; chainid[v] = nchain;

            }

            // buff is the current chain of size k

            nchain++;

        } }

    return tot;

}

void hld(int v) {

    int hd, k; nchain = 0; lvl[0] = -1; dfs(v, 0, 0);

    for(hd = v, k = 0; v != -1; v = cnext[v], k++) {

        head[v] = hd; temp[k] = cost[v];

        chainpos[v] = k; chainid[v] = nchain;

    }

    // buff is the current chain of size k

    nchain++;

}

int lca(int a, int b) {

    while(chainid[a] != chainid[b]) {

if(lvl[head[a]] < lvl[head[b]]) b=parent[head[b]];

   else a = parent[head[a]];

    }

    return (lvl[a] < lvl[b]) ? a : b;

}

1. **Bi-Connected Component :**

/\*G[][]: undirected graph, Separates bi-connected component by edges. \*/

vector< int > G[MAX]; stack< pii > S;

int dfstime; int low[MAX], vis[MAX],used[MAX];

void dfs(int u, int par) {

    int v, i, sz = G[u].size(); pii e, curr;

    used[u] = 1; vis[u] = low[u] = ++dfstime;

    for(i = 0; i < sz; i++) {

        v = G[u][i]; if(v == par) continue;

        if(!used[v]) {

 S.push(pii(u, v)); dfs(v, u);

  if(low[v] >= vis[u]) {

                // new component

curr = pii(u, v);

do { e = S.top(); S.pop();

// e is an edge in current bcc

} while(e != curr); }

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

       }

else if(vis[v] < vis[u]) {

            S.push(pii(u, v));

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

        } } }

1. **2-SAT**

/\*\*\*\*\* Input form of this code is like,

+1 -2

+2 +5

.... \*\*\*\*\*/

vector<int> g[MAX],gt[MAX];

list<int> L; bool vi[MAX]; int num[MAX];

int giveNode(int x,char ch){

x <<=1; if(ch=='-') return x+1; return x;

}

void dfs1(int x){

vi[x] = 1; int i,sz = g[x].size(),y;

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

y = g[x][i]; if(!vi[y])dfs1(y);

}

L.push\_front(x);

}

void dfs2(int x,int cnt){

num[x] = cnt; int i,sz = gt[x].size(),y;

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

y = gt[x][i]; if(num[y]==-1) dfs2(y,cnt);

} }

int main(){

list<int>::iterator it; vector<int> v;

int i,x,y,n,m,t,cs = 1; char ch1,ch2;

scanf("%d",&t);

while(t--){

scanf("%d %d",&m,&n);

for(i = 0;i<2\*n;i++)

g[i].clear(),gt[i].clear(),num[i] = -1,vi[i] = 0;

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

scanf(" %c%d %c%d",&ch1,&x,&ch2,&y);

x = giveNode(x-1,ch1); y = giveNode(y-1,ch2);

g[x^1].pb(y); g[y^1].pb(x);

gt[y].pb(x^1); gt[x].pb(y^1);

}

L.clear();

for(i = 0;i<2\*n;i++) if(!vi[i]) dfs1(i);

int cnt = 0;

for(it=L.begin();it!=L.end();it++)

if(num[\*it]==-1)dfs2(\*it,cnt++);

v.clear();for(i = 0;i<2\*n;i+=2)

if(num[i]==num[i^1]) break;

if(i<2\*n){ puts("No"); continue; }

for(i = 0;i<2\*n;i+=2) if(num[i]>num[i^1])v.push\_back(i/2);

printf("Yes\n%d",v.size());

for(i = 0;i<v.size();i++) printf(" %d",v[i]+1);

puts(""); }

return 0; }

1. **Erdos and Gallai Theorem:**

// Given the degrees of the vertices of a graph, is it possible to construct such // graph Input - the deg[] array

int deg[MM], n, degSum[MM], ind[MM], minVal[MM];

bool ErdosGallai() { //1 indexed

bool poss = true;

int i, sum = 0, j, r;

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

if( deg[i] >= n ) poss = false;

sum += deg[i]; }

//Summation of degrees has to be ODD and all degrees has to be < n - 1

if( !poss || ( sum & 1 ) || ( n == 1 && deg[1] > 0 ) ) return false;

sort( deg + 1, deg + n + 1, greater <int>() );

degSum[0] = 0; j = n;

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

degSum[i] = degSum[i-1] + deg[i]; //CONSTRUCTING: degSum

for( ; j >= 1 && deg[j] < i; j-- ); //CONSTRUCTING: ind

ind[i] = j+1; }

//CONSTRUCTING : minVal

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

j = ind[r]; if( j == n+1 ) minVal[r]=( n - r ) \* r;

else if( j <= r ) minVal[r] = degSum[n] - degSum[r];

else { minVal[r] = degSum[n] - degSum[j-1];

minVal[r] += (j-r-1)\*r; } }

for( r = 1; r < n; r++ ) if( degSum[r] > ( r \* (r-1) + minVal[r] ) ) return false;

return true; }

# MST (Directed Graph):

/\*1. For each node (except the root), look for the minimum weight incoming edge.

2. Look for cycles, if there's no cycle, we already have a tree (which is an MST) goto End

3. Pick one cycle and find an edge p->q, p is in set (not part of the cycle). q is in set (s part of the cycle). Pick this p and q such that: cost of (p->q + sum of all edges in the cycle) - the minimum incoming edge to q (computed in step 1) is minimum. Return to step 2.\*/

struct edge { // Caution: The vertices should be reachable from the root

int v, w;

bool operator < ( const edge &v ) const { return w > v.w; }

};

vector <edge > adj[MAX];// For saving incoming edges and their costs

int DMST( int n, int root ) { // 1 indexed

int i, res=0, pr[MAX],cost[MAX], sub[MAX], sn[MAX], visited[MAX];

vector <int> ::iterator v, it;

vector <int> node[MAX];

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

node[i].clear(); node[i].push\_back( i );

sn[i] = i, sub[i] = pr[i] = 0; }

for(i = 1; i <= n; i++) if( i != root ) {

sort( adj[i].begin(), adj[i].end() ); // sorted in //descending order of w

pr[i] = adj[i].back().v;

cost[i] = sub[i] = adj[i].back().w;

res += cost[i]; }

bool cycle = true;

while( cycle ) {

cycle = false;

memset( visited, 0, sizeof( visited ) );

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

if( visited[i] || sn[i] != i ) continue; int cur = i;

do { visited[cur] = i;cur = pr[cur]; }

while( cur > 0 && !visited[ cur ] );

if( cur > 0 && visited[ cur ] == i ) {

cycle = true;

int start = cur; // assert( sn[start] == start ) ;

do{

if( \*node[cur].begin() != cur ) break;

for( it = node[cur].begin(); it != node[cur].end(); it++) {

sn[ \*it ] = start;

if( cur != start ) node[ start ].push\_back ( \*it );

}

if( cur != start ) node[ cur ].clear();

cur = pr[ cur ]; }

while( cur != start );

int best = INT\_MAX;

for( v = node[start].begin(); v!=node[start].end(); v++) {

while( !adj[\*v].empty() && sn[adj[\*v].back().v] == start)

adj[ \*v ].pop\_back();

if( !adj[\*v].empty() ) {

int tcost = adj[\*v].back().w - sub[ \*v ];

if( tcost < best ) best = tcost, pr[ start ] = adj[\*v].back().v;

} } //assert( best >= 0 && best < INT\_MAX );

cost[ start ] = best;

for( v = node[start].begin(); v != node[start].end(); v++ )sub[\*v] += best;

res += best; } }

for(i = 1; i <= n; i++) pr[i] = sn[ pr[i] ]; }

return res; }

# Weighted Bipartite Matching O(n^3):

// Take input in cost[][]

#define N 55

#define INF 100000000

int cost[N][N], n, max\_match;

int lx[N], ly[N];int xy[N], yx[N];bool S[N], T[N];

int slack[N], slackx[N], prev[N];

void init\_labels() {

memset( lx, 0, sizeof(lx) );

memset( ly, 0, sizeof(ly) );

for( int x = 0; x < n; x++ ) for( int y = 0; y < n; y++ ) lx[x] = max(lx[x], cost[x][y]);

}

void update\_labels() {

int x, y, delta = INF;

for (y = 0; y < n; y++) if (!T[y]) delta = min(delta, slack[y]);

for (x = 0; x < n; x++) if (S[x]) lx[x] -= delta;

for (y = 0; y < n; y++) if (T[y]) ly[y] += delta;

for (y = 0; y < n; y++) if (!T[y]) slack[y] -= delta;

}

void add\_to\_tree( int x, int prevx ) {

S[x] = true; prev[x] = prevx;

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

if (lx[x] + ly[y] - cost[x][y] < slack[y]) {

slack[y] = lx[x] + ly[y] - cost[x][y];

slackx[y] = x;

} }

void augment() {

if( max\_match == n ) return;

int x, y, root;

int q[N], wr = 0, rd = 0;

memset(S, false, sizeof(S));

memset(T, false, sizeof(T));

memset(prev, -1, sizeof(prev));

for( x = 0; x < n; x++ ) if (xy[x] == -1) {

q[wr++] = root = x;

prev[x] = -2;

S[x] = true; break; }

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

slack[y] = lx[root] + ly[y] - cost[root][y];

slackx[y] = root; }

while( true ) {

while( rd < wr ) {

x = q[rd++];

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

if( cost[x][y] == lx[x] + ly[y] && !T[y] ) {

if( yx[y] == -1 ) break;

T[y] = true;

q[wr++] = yx[y];

add\_to\_tree( yx[y], x);

}

if(y < n) break;

}

if(y < n) break;

update\_labels();

wr = rd = 0;

for(y = 0; y < n; y++) if(!T[y] && slack[y] == 0) {

if(yx[y] == -1) {

x = slackx[y]; break; }

else { T[y] = true;

if (!S[yx[y]]) {

q[wr++]=yx[y];

add\_to\_tree(yx[y], slackx[y]);

} } }

if(y < n) break; }

if(y < n) {

max\_match++;

for( int cx = x, cy = y, ty; cx != -2; cx = prev[cx], cy = ty ) {

ty = xy[cx];yx[cy] = cx;

xy[cx] = cy; }

augment(); } }

int hungarian() {

int ret = 0; max\_match = 0;

memset(xy, -1, sizeof(xy));

memset(yx, -1, sizeof(yx));

init\_labels(); augment();

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

return ret; }

# Gomory Hu Tree ( All Pair Max Flow ) O(V \* MaxFlow):

int cap[205][205],flow[205][205],prev[205],w[205],p[205],n;

bool bfs(int s,int t){

int u,i; memo(prev,-1);

queue<int> q; q.push(s); prev[s]=-2;

while(!q.empty()){

u = q.front(); q.pop();

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

if(prev[i]==-1 && cap[u][i]-flow[u][i]>0){

prev[i]= u; q.push(i);

}}}

if(prev[t]==-1) return false;

return true;

}

int dinic(int s,int t){

memo(flow,0); int tot = 0,fl,u,v,i;

while(bfs(s,t)){

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

if(prev[i]==-1 || cap[i][t]-flow[i][t]<=0) continue;

fl = cap[i][t] - flow[i][t];

for(v = i,u=prev[v];v>=0;v =u,u=prev[v]) fl = min(fl,cap[u][v]-flow[u][v]);

for(v = i,u=prev[v];v>=0;v =u,u=prev[v]) {

flow[u][v]+=fl,flow[v][u]-=fl;

}

flow[i][t]+=fl; flow[t][i]-=fl;

tot+=fl; } }

return tot;

}

int main(){

int t,cs,i,j,source,sink,fl,k,inf=1<<29;

scanf("%d",&t);

for(cs=1;cs<=t;cs++){

scanf("%d",&n);

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

p[i]=0;

for(j=0;j<n;j++) scanf("%d",&cap[i][j]); }

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

sink = p[source];

fl = dinic(source,sink);

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

if(i==source || prev[i]==-1 || p[i]!=sink) continue;

p[i]=source; }

w[source]=fl;

if(prev[p[sink]]==-1) continue;

p[source]=p[sink];

p[sink]=source; w[source]=w[sink];

w[sink]=fl; }

for(i=0;i<n;i++) for(j=0;j<n;j++) cap[i][j]=0;

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

if(p[i]==i) continue;

cap[i][p[i]]=cap[p[i]][i]=w[i];

}

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

cap[i][j]=max(cap[i][j],min(cap[i][k],cap[k][j]));

for(i=0;i<n;i++) cap[i][i]=0;

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

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

if(j) printf(" ");

printf("%d",cap[i][j]);

} puts(""); } }

return 0;

}

**6. Euler Circuit:**

list< int> cyc;

voideuler( list< int>::iterator i, intu ) {

int v;

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

if( mat[u][v] ) {

mat[u][v]--; mat[v][u]--;

euler( cyc.insert( i, u ) , v );

} } }

////

cyc.clear();

euler( cyc.begin(), 1 );

list<int>::iterator iter;

for( i=0,iter = cyc.begin(); iter != cyc.end(); iter++,i++ )

ans[i] = node[\*iter];

////

**7. Articulation Point:**

vector< int > G[MAX];

int low[MAX], vis[MAX], used[MAX], cut[MAX], dfstime;

void dfs(int u, int par = -1) {

    int i, v, child = 0;

    used[u] = 1;

    vis[u] = low[u] = ++dfstime;

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

        v = G[u][i];

        if(v == par) continue;

if(used[v]) low[u] = min(low[u], vis[v]);

        else {

            child++; dfs(v, u);

  low[u] = min(low[u], low[v]);

if(low[v] >= vis[u]) cut[u] = 1;

        }

    }

    if(par == -1) cut[u] = (child > 1);

}

**String:**

1. **Suffix Array (nlg(n)):**

/\* Suffix array implementation using bucket sorting + lcp. Complexity O(n log n), str[] is the target string,

n is its length and suffix[i] contains i'th sorted suffix position.\*/

const int MAXN = 1 << 16;

const int MAXL = 16;

int n, stp, mv, suffix[MAXN], tmp[MAXN];

int sum[MAXN], cnt[MAXN], rank[MAXL][MAXN];char str[MAXN];

inline bool equal(const int &u, const int &v){

    if(!stp) return str[u] == str[v];

    if(rank[stp-1][u] != rank[stp-1][v]) return false;

int a = u + mv < n ? rank[stp-1][u+mv] : -1;

 int b = v + mv < n ? rank[stp-1][v+mv] : -1;

    return a == b;

}

void update(){

    int i, rnk;

for(i =0;i< n; i++)sum[i]=0;

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

        suffix[i] = tmp[i];

        if(i && !equal(suffix[i], suffix[i-1])) {

rank[stp][suffix[i]] = ++rnk;

sum[rnk+1] = sum[rnk];

        } else rank[stp][suffix[i]] = rnk;

        sum[rnk+1]++;

    } }

void Sort() {

    int i;

for(i = 0; i < n; i++) cnt[i] = 0;

memset(tmp, -1, sizeof tmp);

    for(i = 0; i < mv; i++){

 int idx=rank[stp-1][n-i-1];

        int x = sum[idx];

        tmp[x + cnt[idx]] = n - i - 1; cnt[idx]++;

    }

    for(i = 0; i < n; i++){

   int idx = suffix[i] - mv;

        if(idx < 0)continue;

        idx = rank[stp-1][idx]; int x = sum[idx];

tmp[x + cnt[idx]] = suffix[i] - mv; cnt[idx]++;

    } update(); return;

}

inline bool cmp(const int &a, const int &b){

    if(str[a]!=str[b]) return str[a]<str[b];

    return false;

}

void SortSuffix() {

    int i;

for(i = 0;i< n; i++) tmp[i] = i;

    sort(tmp, tmp + n, cmp);

stp = 0; update(); ++stp;

for(mv = 1; mv < n; mv <<= 1) {

        Sort(); stp++;

    }

    stp--;

    for(i = 0; i <= stp; i++) rank[i][n] = -1;

}

inline int lcp(int u, int v) {

    if(u == v) return n - u; int ret, i;

    for(ret = 0, i = stp; i >= 0; i--) {

if(rank[i][u] == rank[i][v]) {

ret += 1<<i;u += 1<<i;v += 1<<i;

        } }

    return ret;

}

1. **Aho Corasic :**

int tree[250010][26] ;

int dp[250010] ;

int par[250010] ;

int pre[250010] ;

char val[250010] ;

int clr[250010][2] ;

int node,l ;

char in[510] ;

int end[510] ;

char S[1000010] ;

VI depth[510] ;

void ins(int pos) {

int cur = 0,i ;

for(i=0;in[i];i++) {

if(tree[cur][in[i]-'a']) cur = tree[cur][in[i]-'a'] ;

else{

clr[l][0] = cur ; clr[l++][1] = in[i]-'a' ;

par[node+1] = cur ;

cur = tree[cur][in[i]-'a'] = ++node ;

val[node] = in[i]-'a' ; depth[i].pb(node) ;

} }

end[pos] = cur ;

}

int prefixCalc(int u){

if(par[u]==0) return 0 ;

int k = pre[par[u]] ;

while(1) {

if(tree[k][val[u]]) return tree[k][val[u]] ;

if(k==0) return 0 ; k = pre[k] ;

} }

void freqCalc(char \*in) {

int k = 0,i ;

for(i=0;in[i];i++) {

while(1) {

if(tree[k][in[i]-'a']) {

k = tree[k][in[i]-'a'] ; break ;

}

if(k==0) break ;

k = pre[k] ; }

dp[k] ++ ;

} }

int main() {

int n,T,t=1,m,i,j,k;

//freopen("in.txt","r",stdin);

//freopen("out.txt","w",stdout);

scanf("%d",&T); for(t=1;t<=T;t++) //while(scanf("%d",&n)==1){

rep(i,0,502) depth[i].clear() ; SD(n) ; SS(S) ; l = node = 0 ;

rep(i,0,n) {

scanf("%s",in) ; ins(i) ; }

rep(i,0,502) rep(j,0,sz(depth[i]))

pre[depth[i][j]]=prefixCalc(depth[i][j]) ;

freqCalc(S) ;

rem(i,501,-1) {

rep(j,0,sz(depth[i])) {

dp[pre[depth[i][j]]] += dp[depth[i][j]] ;

} }

rep(i,0,n) printf("%d\n",dp[end[i]]) ;

rep(i,0,node+1) dp[i] = 0 ;

rep(i,0,l) tree[clr[i][0]][clr[i][1]] = 0 ;

}

return 0;

}

1. **Minimum Rotation :**

inline int minimumExpression(char \*s) {

int i, j, k, n, len, p, q;

len = n = strlen(s), n <<= 1, i = 0, j = 1, k = 0;

while(i + k < n && j + k < n) {

p = i+k >= len ? s[i+k-len] : s[i+k]; q = j+k >= len ? s[j+k-len] : s[j+k];

    if(p == q) k++;

else if(p > q) { i = i+k+1; if(i <= j) i = j+1; k = 0; }

else if(p < q) { j = j+k+1; if(j <= i) j = i+1; k = 0; }

    }

    return i < j ? i : j;

}

1. **Manakera’s Algorithm :**

For the case subpalindromes odd length, ie for calculating the array d_1 [],

vector<int> d1 (n);

int l=0, r=-1;

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

int k = (i>r ? 0 : min (d1[l+r-i], r-i)) + 1;

while (i+k < n && i-k >= 0 && s[i+k] == s[i-k]) ++k;

d1[i] = k--;

if (i+k > r)

l = i-k, r = i+k;

}

For subpalindromes even length, ie for calculating the array d_2 [],

vector<int> d2 (n);

l=0, r=-1;

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

int k = (i>r ? 0 : min (d2[l+r-i+1], r-i+1)) + 1;

while (i+k-1 < n && i-k >= 0 && s[i+k-1] == s[i-k]) ++k;

d2[i] = --k;

if (i+k-1 > r)

l = i-k, r = i+k-1;

}

1. **KMP :**

void prefixFunc(int \*P, int lp){

pre[0] = 0 ; int i,k ;

rep(i,1,lp) { k = pre[i-1] ;

while(1) { if(P[i]==P[k]) break ; if(k==0) {

k-- ; break ; } k = pre[k-1] ; }pre[i] = k + 1 ; } }

int kmp(int \*A, int la, int \*B, int lb) // search // for B in A

{

prefixFunc(B,lb) ; int k = 0, i=0, ret=0 ;

if(A[0]==B[0]) k ++ ;

if(k==lb) ret ++ ;

rep(i,1,la) {

while(1) { if(A[i]==B[k]) break ;

if(k==0) { k -- ; break ; }

k = pre[k-1] ; }

k ++ ; if(k==lb) ret ++ ;

}

return ret ;

}

1. **Suffix Autimata :**

#define MAXLEN 1000006 //REMEMBER TO TAKE DOUBLE SIZED ARRAY

struct State{

int len, link; map < char , int > nxtc;

} SA[2\*MAXLEN];

int FirstPos[2\*MAXLEN], DistSub[2\*MAXLEN], NumOcc[2\*MAXLEN];

bool vis[2\*MAXLEN];

char str[MAXLEN];

vector < pair < int , int > > lenState;

class SuffixAutomata {

int sz, last; int fnl;

void init(int idx) {

SA[idx].len=0,SA[idx].link = -1;

FirstPos[idx] = -1;

SA[idx].nxtc.clear();

}

bool traverse(int cur,char \*ch){

if (\*ch == 0) {printf("first occurance lasts at = %d :: total # of occurances = %d :: node = %d\nlink = %d :: len = %d\n", FirstPos[cur], NumOcc[cur], cur, SA[cur].link, SA[cur].len); return true; }

if (SA[cur].nxtc.count(\*ch) == 0) return false;

int p = SA[cur].nxtc[\*ch];

return traverse(p, ++ch);

}

int distSubs(int cur){if (vis[cur]) return DistSub[cur];

vis[cur] = true;

if (cur) DistSub[cur] = 1;

else DistSub[cur] = 0;

map < char , int > :: iterator im; map < char , int > &M = SA[cur].nxtc;

for (im = M.begin(); im != M.end(); ++im) DistSub[cur] += distSubs(im->second);

return DistSub[cur]; }

public:

SuffixAutomata(){

sz = last = 0; init(0);

lenState.clear();++sz;

}

int size() {return sz;}

void extend(char ch,int idx=-1){

int p, q, clone, cur = sz++;

fnl = cur; init(cur);

SA[cur].len = SA[last].len + 1;

FirstPos[cur] = idx;

NumOcc[cur] = 1;

lenState.push\_back(make\_pair(SA[cur].len, cur));

for (p = last; p != -1 && SA[p].nxtc.count(ch) == 0; p = SA[p].link) SA[p].nxtc[ch] = cur;

if (p == -1) SA[cur].link = 0;

else{ q = SA[p].nxtc[ch];

if (SA[p].len + 1 == SA[q].len) SA[cur].link = q;

else { clone = sz++;

init(clone); SA[clone] = SA[q];

SA[clone].len = SA[p].len + 1;

for (; p != -1 && SA[p].nxtc[ch] == q; p = SA[p].link) SA[p].nxtc[ch] = clone;

SA[cur].link = SA[q].link = clone; FirstPos[clone] = FirstPos[q]; NumOcc[clone] = 0; lenState.push\_back(make\_pair(SA[clone].len, clone)); }}

last = cur; }

void traverse(char \*str) {

if (traverse(0, str) == 0) puts("not a substring"); }

int distSubs() {

int i; memset(vis, false, sizeof(vis)); sort(lenState.begin(),lenState.end());

for (i=lenState.size()-1; i>=0; --i) distSubs(lenState[i].second);

return distSubs(0); }

void numOcc() {

int i,p,q;sort(lenState.begin(), lenState.end());for(i=lenState.size()-1; i>=0; --i){

q = lenState[i].second; p = SA[q].link; if (p<1) continue;

NumOcc[p] += NumOcc[q]; } } };

/\*sa.extend(character, index in string) diye automata te character jog kora jabe

shob char jog kora shesh hole, sa.distSubs() call korle return korbe number of substrings (distinct)

shob char jog kora shesh hole, sa.numOcc() call korle, prottekta state er respect e, oi state ta kotobar string e occur kore sheita NumOcc[state] e saved hobe, jamon , abcbcabc er jonno, NumOcc["abc"] = 2, NumOcc["bc"] = NumOcc["c"] = 3 etc, \*\*string diye obosshoi access kora jabe nah, oita bujha'r shubidha'r jonno :)extend kora'r shomoy index jodi pass kora hoy, tahole FirstPos[state] diye oi state ta prothom je position e occur kore sheita paowa jabe, SA[state].len diye oi state er length o paowa jabe, jodi FirstPos er jaygay LastPos lage, then obviously string ta ulta kore automata te save korlei hobe, and len - n + 1 type calculation :)

traverse(string) diye traverse kora jabe, private traverse function er bhitre print e details ache :) \*/

int main() {

int i;

while(scanf("%s", str) != EOF){

SuffixAutomata temp;

for (i=0; str[i]; ++i){

temp.extend(str[i], i); /\* index deyata optional, substring er first occurance lagle eikhane index pass korte hobe for each entry\*/ } }

return 0;

}

**Geometry :**

1. **Convex Hull :**

bool com(point a, point b) {

long long d = area(P0, a, b);

if(d<0) return false;

if(d==0 && dist(P0, b) < dist(P0, a)) return false; return true;

}

int convexhull(int np) //gets num of //points,returns num of elements on convex hull { //intermediate points on same line //are discarded

int i,nc,pos = 0;

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

if(P[i].y<P[pos].y || (P[i].y==P[pos].y && P[i].x<P[pos].x))

pos = i; swap(P[0], P[pos]);

P0 = P[0];

stable\_sort(P+1, P+np, com);

C[0] = P[0]; if(np>=2) C[1] = P[1];

for(i=nc=2; i<np; i++) {

while(nc>=2 && area(C[nc-2], C[nc-1], P[i]) <= 0) nc--;

C[nc++] = P[i];

}

if(np==1) nc=1;

else if(nc==2)

if(P[0].x == P[1].x && P[0].y == P[1].y) nc=1;

return nc;

}

1. **fitting a rectangle inside another:**

//(a,b) fits in (c,d)?

bool fits( int a, int b, int c, int d ) {

double X, Y, L, K, DMax;

if( a < b ) swap( a, b );

if( c < d ) swap( c, d );

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

if( d >= b ) return false;

X = sqrt( a\*a + b\*b );

Y = sqrt( c\*c + d\*d );

if( Y < b ) return true;

if( Y > X ) return false;

L = ( b - sqrt( Y\*Y - a\*a) ) /2;

K = ( a - sqrt( Y\*Y - b\*b) ) /2;

DMax = sqrt(L \* L + K \* K);

if( d >= DMax ) return false;

return true;

}

# Mirror point(mx,my) of a point(x,y) w.r.to a line(ax+by+c=0):

void mirrorPoint(double a,double b,double c,double x,double y,double &mx,double &my) {

mx = - x\*(a\*a-b\*b) - 2.0\*a\*b\*y - 2.0\*a\*c; mx /= (a\*a+b\*b);

my = y\*(a\*a-b\*b) - 2.0\*a\*b\*x - 2.0\*b\*c; my /= (a\*a+b\*b);

}

1. **Points/radius given, find circle:**

/\* **Circle through 3 points**

The 3 points are (x[0], y[0]), (x[1], y[1]) and (x[2], y[2]). The centre of the circle is returned as (r[0], r[1]). \*/

double circle3pts( double x[], double y[], double r[] ) {

double lix[4], liy[4];

lix[0] = 0.5 \* ( x[0] + x[1] ); liy[0] = 0.5 \* ( y[0] + y[1] );

lix[1] = lix[0] + y[1] - y[0]; liy[1] = liy[0] + x[0] - x[1];

lix[2] = 0.5 \* ( x[1] + x[2] ); liy[2] = 0.5 \* ( y[1] + y[2] );

lix[3] = lix[2] + y[2] - y[1]; liy[3] = liy[2] + x[1] - x[2];

if( !lineIntersect( lix, liy, r ) ) return -1.0;

return sqrt( ( r[0] - x[0] ) \* ( r[0] - x[0] ) +

( r[1] - y[0] ) \* ( r[1] - y[0] ) ); }

/\* **Circle of a given radius through 2 points**

The returned center is never to the right of the vector \*/

bool circle2ptsRad( double x1, double y1, double x2, double y2, double r, double ctr[2] ) {

double d2 = ( x1 - x2 ) \* ( x1 - x2 ) + ( y1 - y2 ) \* ( y1 - y2 );

double det = r \* r / d2 - 0.25;

if( det < 0.0 ) return false;

double h = sqrt( det );

ctr[0] = ( x1 + x2 ) \* 0.5 + ( y1 - y2 ) \* h;

ctr[1] = ( y1 + y2 ) \* 0.5 + ( x2 - x1 ) \* h;

return true;

}

1. **Closest Pair Problem:**

point p[MM], s1[MM], s2[MM];

bool sortX(point &a, point &b) { return ( a.x == b.x ) ? a.y < b.y : a.x < b.x; }

bool sortY(point &a, point &b) { return ( a.y == b.y ) ? a.x < b.x : a.y < b.y; }

double closestPair( int k1, int k2 ){

double d, d2 ,d3;

if(k2-k1+1 == 1) return 0;

if(k2-k1+1 == 2) return Distance(p[k1], p[k2]);

if(k2-k1+1 == 3) {

d = Distance( p[k1], p[k1+1] );

d2 = Distance( p[k1+1], p[k1+2]);

d3 = Distance( p[k1+2], p[k1]);

return min( min(d, d2), d3 );

}

int k, i, j, ns1, ns2;

k = (k1 + k2) / 2;

d = closestPair(k1 , k);

d2 = closestPair(k+1 , k2);

if(d > d2)d = d2;ns1 = ns2 = 0;

for(i = k; i>=k1 ; i--) {

if( p[k].x - p[i].x > d ) break;

s1[ ns1++ ] = p[i];

}

sort(s1, s1+ns1, sortY);

for(i = k+1; i<=k2 ; i++) {

if( p[i].x - p[k].x > d ) break;

s2[ ns2++ ] = p[i]; }

sort(s2, s2+ns2, sortY);

for(i=0;i<ns1;i++) {

for(j=0;j<ns2;j++) {

if(s2[j].y - s1[i].y > d) break;

d = min( d, Distance( s1[i], s2[j] ) );

} }

return d;

}

/\* ….sort( p, p+n, sortX ); double d = closestPair(0,n-1); ….\*/

1. **Polar Angle :**

/\*Returns an angle in the range [0, 2\*Pi) of a given Cartesian point. If the point is (0,0), -1.0 is returned.\*/

double polarAngle( P p ) {

if( fabs( p.x ) <= EPS && fabs( p.y ) <= EPS ) return -1.0;

if( fabs( p.x ) <= EPS ) return ( p.y > EPS ? 1.0 : 3.0 ) \* acos( 0 );

double theta = atan( 1.0 \* p.y / p.x );

if( p.x > EPS ) return( p.y >= -EPS ? theta : ( 4 \* acos( 0 ) + theta ) );

return( 2 \* acos( 0 ) + theta );

}

1. **Point In Polygon :**

bool pointInPoly( P p, vector< P > &poly ) {

int n = poly.size();

double ang = 0.0;

for(int i=n-1,j=0;j<n;i=j++ ){

P v( poly[i].x - p.x, poly[i].y - p.y );

P w( poly[j].x - p.x, poly[j].y - p.y );

double va = polarAngle( v );

double wa = polarAngle( w );

double xx = wa - va;

if( va < -0.5 || wa < -0.5 || fabs( fabs( xx ) - 2 \* acos( 0 ) ) < EPS ) {

// POINT IS ON THE EDGE

ang += 2 \* acos( 0 );

continue; }

if( xx < -2 \* acos( 0 ) ) ang += xx + 4 \* acos( 0 );

else if( xx > 2 \* acos( 0 ) ) ang += xx - 4 \* acos( 0 );

else ang += xx;

}

return( ang \* ang > 1.0 );

}

1. **Triangle Area From 3 medians length :**

double triAreaFromMedians( double ma, double mb, double mc ) {

double x = 0.5 \* ( ma + mb + mc );

double a = x \* ( x - ma ) \* ( x - mb ) \* ( x - mc );

if( a < 0.0 ) return -1.0;

else return sqrt( a ) \* 4.0 / 3.0;

}

1. **Distance from a point to a line :**

// Does not work for degenerate lines (when answer is undefined).

double distToLine(double ax, double ay,double bx, double by,double px, double py,double \*cpx, double \*cpy ){

//Formula: cp = a + (p-a).(b-a) / |b-a| \* (b-a)

double proj = ( ( px - ax ) \* ( bx - ax ) + ( py - ay ) \* ( by - ay )) / ( ( bx - ax ) \* ( bx - ax )+( by - ay ) \* ( by - ay ));

\*cpx = ax + proj \* ( bx - ax );

\*cpy = ay + proj \* ( by - ay );

return dist( px, py,\*cpx,\*cpy );

}

1. **Destance from a point to a line segment :**

double distToLineSegment(

double ax, double ay, double bx, double by, double px, double py, double \*cpx, double \*cpy ) {

if( ( bx - ax ) \* ( px - ax ) + ( by - ay ) \* ( py - ay ) < EPS ) {

\*cpx = ax; \*cpy = ay;

return dist( ax, ay, px, py );

}

if( ( ax - bx ) \* ( px - bx ) + ( ay - by ) \* ( py - by ) < EPS ) { \*cpx = bx; \*cpy = by;

return dist( bx, by, px, py );

}

return distToLine( ax, ay, bx, by, px, py, cpx, cpy );

}

1. **Point In Convex Polygon :**

/\* C[] array of points of convex polygon in ccw order, nc number of points in C, p target points.

returns true if p is inside C (including edge) or false otherwise. complexity O(lg n)\*/

inline bool inConvexPoly(point \*C, int nc, const point &p) {

  int st = 1, en = nc - 1, mid;

    while(en - st > 1) {

        mid = (st + en)>>1;

if(triArea2(C[0], C[mid], p) < 0) en = mid; else st = mid; }

if(triArea2(C[0], C[st], p) < 0) return false;

    if(triArea2(C[st], C[en], p) < 0) return false;

    if(triArea2(C[en], C[0], p) < 0) return false; return true;

}

1. **Segment-Segment Intersection :**

template< class T > bool lineSegIntersect( vector< T > &x, vector< T > &y ) {

double ucrossv1 = ( x[1] - x[0] ) \* ( y[2] - y[0] ) - ( y[1] - y[0] ) \* ( x[2] - x[0] );

double ucrossv2 = ( x[1] - x[0] ) \* ( y[3] - y[0] ) - ( y[1] - y[0] ) \* ( x[3] - x[0] );

if( ucrossv1 \* ucrossv2 > 0 ) return false;

double vcrossu1 = ( x[3] - x[2] ) \* ( y[0] - y[2] ) - ( y[3] - y[2] ) \* ( x[0] - x[2] );

double vcrossu2 = ( x[3] - x[2] ) \* ( y[1] - y[2] ) - ( y[3] - y[2] ) \* ( x[1] - x[2] );

return( vcrossu1 \* vcrossu2 <= 0 ); }

/\*P1, p2 makes first segment, p3, p4 makes the second segment\*/

inline bool intersect(const Point &p1, const Point &p2, const Point &p3, const Point &p4) {

    i64 d1, d2, d3, d4;

    d1 = direction(p3, p4, p1);d2 = direction(p3, p4, p2);d3 = direction(p1, p2, p3);

    d4 = direction(p1, p2, p4);

    if(((d1 < 0 && d2 > 0) || (d1 > 0 && d2 < 0)) && ((d3 < 0 && d4 > 0) || (d3 > 0 && d4 < 0))) return true;

    if(!d3 && onsegment(p1, p2, p3)) return true;

    if(!d4 && onsegment(p1, p2, p4)) return true;

    if(!d1 && onsegment(p3, p4, p1)) return true;

    if(!d2 && onsegment(p3, p4, p2)) return true;

    return false; }

1. **Line-Circle Intersection:**

inline bool intersection(circle C,line L,point &p1,point &p2) {

if( Distance( C.center, L ) > C.r + eps ) return false;

double a, b, c, d, x = C.center.x, y = C.center.y;

d = C.r\*C.r - x\*x - y\*y;

if( eq( L.a, 0) ) {

p1.y = p2.y = -L.c / L.b;

a = 1; b = 2 \* x;

c = p1.y \* p1.y - 2 \* p1.y \* y - d;

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

d = sqrt( fabs (d) );

p1.x = ( b + d ) / ( 2 \* a );

p2.x = ( b - d ) / ( 2 \* a ); }

else {

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

b = 2 \* ( L.a \* L.a \* y - L.b \* L.c - L.a \* L.b \* x);

c = L.c \* L.c + 2 \* L.a \* L.c \* x - L.a \* L.a \* d;

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

d = sqrt( fabs(d) );

p1.y = ( b + d ) / ( 2 \* a );

p2.y = ( b - d ) / ( 2 \* a );

p1.x = ( -L.b \* p1.y -L.c ) / L.a;

p2.x = ( -L.b \* p2.y -L.c ) / L.a; } return true; }

1. **Intersection Area Between Two Circles :**

double area\_and(Circle p,Circle q){ double d,h,s,ta,pt,qt,ret;

if( p.rad > q.rad )swap(p,q);

d = D1(p.cen,q.cen);

if( d < q.rad - p.rad || Z(d) || E(d,q.rad - p.rad))

return pi\*S(p.rad);

if( d > p.rad + q.rad || E(d,p.rad + q.rad)) return0;

s = (p.rad + q.rad + d) / 2;

ta = sqrt(s\*(s-d))\*sqrt((s-p.rad)\*(s-q.rad));

h = 2\*ta / d;

pt = acos( (S(p.rad) + D2(p.cen,q.cen) - S(q.rad)) / (2.\*p.rad\*d) );

qt = acos( (S(q.rad) + D2(p.cen,q.cen) - S(p.rad)) / (2.\*q.rad\*d) );

return= pt\*S(p.rad) + qt\*S(q.rad) - h\*d;

}

1. **Angle Between Vectors :**

inline double angleBetweenVectors( point O, point A, point B ) { // vector // OA to OB

point t1, t2;

t1.x = A.x - O.x; t1.y = A.y - O.y;

t2.x = B.x - O.x; t2.y = B.y - O.y;

double theta = (atan2(t2.y, t2.x) - atan2(t1.y, t1.x));

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

return theta;

}

1. **Determining if a point lies on the interior of a 3D convex polygon:**

// To determine whether a point is on the interior of a convex polygon in 3D, one

// might be tempted to first determine whether the point is on the plane, then

// determine its interior status. Both of these can be accomplished at once by

// computing the sum of the angles between the test point (q below) and every pair of

// edge points p[i]->p[i+1]. This sum will only be twopi if both the point is on the

// plane of the polygon AND on the interior. The angle sum will tend to 0 the further

// away from the polygon point q becomes. The following code snippet returns the angle

// sum between the test point q and all the vertex pairs. The angle sum is in radians.

#define EPSILON 0.0000001

#define MODULUS(p) (sqrt(p.x\*p.x + p.y\*p.y + p.z\*p.z))

const double TWOPI = 6.283185307179586476925287, RTOD = 57.2957795;

double CalcAngleSum( point3D q, point3D \*p, int n ) {

double m1,m2,anglesum=0,costheta;

point3D p1, p2;

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

p1.x = p[i].x - q.x; p1.y = p[i].y - q.y; p1.z = p[i].z - q.z;

p2.x = p[(i+1)%n].x - q.x;

p2.y = p[(i+1)%n].y - q.y;

p2.z = p[(i+1)%n].z - q.z;

m1 = MODULUS(p1), m2 = MODULUS(p2);

if(m1\*m2 <= EPSILON) return(TWOPI); // We are on a node, consider this inside

else costheta = (p1.x\*p2.x + p1.y\*p2.y + p1.z\*p2.z) / (m1\*m2);

anglesum += acos(costheta); }

return(anglesum); }

1. **Centroid:**

Sum of(xi+xi+1)(xiyi+1–yixi+1) for x for all i divided by 6A. y is similar.

1. **Centroid of a 3D shell described by 3 vertex facets:**

*The centroid C of a 3D object made up of a collection of N triangular faces with*

*vertices* ***(ai,bi,ci)*** *is given below.* ***Ri*** *is the average of the vertices of the i'th*

*face and Ai is twice the area of the i'th face. Note the faces are assumed to be*

*thin sheets of uniform mass, they need not be connected or form a solid object.*



# Minimal enclosing circle given some points:

struct Point{

LD x,y;

Point(){}

Point(double a, double b){x = a, y = b;}

bool operator <(Point b)const{

if(!eq(x,b.x) )return x < b.x;

return y < b.y;

}

bool operator == (Point b) const{

if(eq(x,b.x) && eq(y,b.y)) return true; return false; } };

double ang(Point a,Point b,Point c){ //returns angle <bac

double absq =sq\_distance(a , b);

double bcsq = sq\_distance(c , b), acsq = sq\_distance(a , c);

double cosp = (absq+acsq - bcsq)/(2.0\*sqrt(absq \* acsq) ); return acos(cosp); }

struct side{

Point a,b;

side(){}

side(Point aa,Point bb){ a = aa,b = bb;}

};

struct circle{

Point center; double r; };

int main(){

int tc, i; cin>>tc;

while(tc--){

P.clear(); int n; double a,b;

cin >> n; while(n -- )

scanf("%lf %lf",&a,&b), P.push\_back(Point(a,b));

ConvexHull(); circle res;

side now(C[0],C[1]);

int ai = 0, aj = 1;

while(true) {

double tmp, mn = 1e10;

int rem;

FOR(i,nC){

if(i!= ai && i!=aj){

tmp = ang(C[i],C[ai],C[aj]);

if(tmp < mn) mn =tmp,rem = i;} }

if( 2\*mn >= PI){

res.r = sqrt(sq\_distance(C[ai], C[aj]))/2.0;

res.center.x = (C[ai].x + C[aj].x)/2;

res.center.y = (C[ai].y + C[aj].y)/2;

break; }

double a1 = ang(C[ai],C[aj],C[rem]);

double a2 = ang(C[aj],C[ai],C[rem]);

double a3 = ang(C[rem],C[ai],C[aj]);

if(a1<=RA && a2 <=RA && a3<=RA){

res=CircleThrough3Points(C[ai],C[aj],C[rem]); break; }

else if(a1 > RA)ai = aj,aj =rem;

else if(a2 >RA)ai = ai,aj = rem; }

printf("%.2lf\n%.2lf %.2lf\n",res.r,res.center.x,res.center.y); }

return 0; }

# Circle Union Area - O(n3logn):

// slicing + interval mainpulation, n^2 \* nlogn = n^3logn

#define MAX 102

#define EPS 1e-9

double pi = acos(-1.);

struct Circle{

double x,y,r,xlo,xhi;

Circle(){}

Circle(double a, double b, double c){x = a, y = b, r = c; xlo=x-r; xhi=x+r; }

}tmp;

int n; Circle c[MAX];

#define OPEN 0

#define CLOSE 1

struct Event{

int type;

double y1,y2,aa; //aa = arcarea of the event's host circle

Event(){}

Event(int t,double yy1,double yy2,double aaa){

type = t,y1=yy1, y2 =yy2,aa=aaa;

} };

bool operator <(const Event &p,const Event &q){ //event sort function

double py = p.y1 + p.y2, qy = q.y1 + q.y2;

if(fabs(py-qy) < EPS) return p.type < q.type;

return py > qy + EPS;

}

//this is enough as, p.y1 > q.y1 //<=> p.y2 > q.y2 (slicing)

//circles MUST intersect. returns only the 'x' of intersections

double mysqrt(double s){

if(s<EPS) return 0;

return sqrt(s); }

double D1( Circle &c1, Circle &c2 ){ return mysqrt( ( c1.x - c2.x ) \* ( c1.x - c2.x ) + ( c1.y - c2.y ) \* ( c1.y - c2.y ) ); }

double D2( Circle &c1, Circle &c2 ){

return ( ( c1.x - c2.x ) \* ( c1.x - c2.x ) + ( c1.y - c2.y ) \* ( c1.y - c2.y ) ); }

double S( double a ) { return a \* a ; }

bool getCircleIsects(Circle c1,Circle c2,double &x1,double &x2){

if(c1.r +EPS< c2.r)swap(c1,c2);

double d = D1(c1,c2);

double a = ( S(c1.r) + S(d) - S(c2.r) )/(2.\*d);

double h = mysqrt( S(c1.r) - S(a) );

x1 = c1.x + (a\*(c2.x-c1.x) - h\*(c2.y-c1.y))/d;

x2 = c1.x + (a\*(c2.x-c1.x) + h\*(c2.y-c1.y))/d;

return true; }

double res;

vector<double> vx, X; //for //slicing

int ne, tos;

Event e[2\*MAX]; //events of arc //open/close

Event Stack[MAX+1]; //Stack - remember earlier openings

double inarea[MAX+1];//Stack - calculate the area that is covered by inner circles

double myacos( double a ){

if( fabs( a + 1 )<EPS)return pi;

if( fabs( a - 1 ) <EPS)return 0;

return acos( a );

}

double arcarea( double C, double D, Circle &c ){

double AOC, BOD, AOB, O = c.x;

if( C > O + EPS && D > O + EPS ) swap( C, D );

double OC = fabs( O - C ), OD = fabs( O - D );

AOC = myacos( OC / c.r ), BOD = myacos( OD / c.r );

if(( C + EPS < O && D + EPS < O ) || ( C > O + EPS && D > O + EPS ) );

else BOD = pi - BOD;

AOB = ( BOD - AOC ) / 2;

return c.r \* c.r \*( AOB - cos(AOB)\*sin(AOB) );

}

bool comp(const double &aa,const double &bb){return aa+EPS <bb; }

void circleUnionArea(int n,Circle \*c){

int i,j,k; double d,x1,x2;

// << slicing starts >>

vx.clear();

for(i=0;i<n;i++){ //no need for center.x

vx.push\_back(c[i].x - c[i].r);

vx.push\_back(c[i].x + c[i].r); }

//insert all possible x[intersections]

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

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

d = D1(c[i],c[j]);

double dd2 = D2(c[i],c[j]);

double ss1 = S(c[i].r + c[j].r);

double ss2 = S(c[i].r - c[j].r);

if(dd2 > ss1 + EPS||fabs(dd2 - ss1)<EPS || dd2 + EPS < ss2 || fabs(dd2 - ss2)<EPS)continue;

if(getCircleIsects(c[i],c[j],x1,x2) ) vx.push\_back(x1), vx.push\_back(x2);

}

sort(vx.begin(), vx.end(),comp); X.clear(); X.push\_back(vx[0]);

for(i=1;i<vx.size();i++)

if( fabs(vx[ i - 1 ] - vx[ i ] ) > EPS ) X.push\_back(vx[i]);

// << slcing end >>

double area,xgap, OC, AC, OD,BD;

for(k=0;k+1<X.size();k++){ //for each X slice

x1 = X[k]; x2 = X[k+1]; xgap = x2-x1; ne = 0;

for(i=0;i<n;i++){ // 2 events //for each circle

if(x2 + EPS < c[i].xlo || fabs(x2 - c[i].xlo)<EPS) continue;

if(c[i].xhi +EPS < x1 || fabs(x1 - c[i].xhi)<EPS) continue;

OC = (x1-c[i].x);

AC = mysqrt( S(c[i].r) -S(OC) );

OD = (x2-c[i].x);

BD = mysqrt( S(c[i].r) -S(OD) );

area = arcarea(x1,x2,c[i]);

e[ne++] = Event(OPEN, c[i].y+AC, c[i].y+BD, area);

e[ne++] = Event(CLOSE,c[i].y-AC, c[i].y-BD, area); }

if(ne==0)continue;

sort(e,e+ne);

for(i=0;i<=n;i++)inarea[i] = 0; //init the inner area sum tos = 0; for(i=0;i<ne;i++) if(e[i].type == CLOSE){

area = Stack[tos-1].aa + e[i].aa;

area += 0.5\*xgap\*((Stack[tos-1].y1 - e[i].y1)+(Stack[tos-1].y2 - e[i].y2));

res += area - inarea[tos - 1];

if(tos>=2)inarea[tos-2] += area;

inarea[tos-1] = 0; tos--; }

else Stack[tos++] = e[i]; } }

int main(){

int i, CC;

while(scanf("%d",&n) == 1 && n){

CC = 0;

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

scanf("%lf%lf%lf",&tmp.x,&tmp.y,&tmp.r);

if(tmp.r<EPS) continue;

c[CC++ ] =Circle(tmp.x,tmp.y,tmp.r ); }

res = 0;

circleUnionArea( CC, c );

printf("%.3lf\n",res + EPS ); }

return 0;

}

1. **Point Classification:**

/\* p2 on which side of p0(origin)-p1(destination) \*/

int classify(Point p0,Point p1,Point p2){

Point a,b; double t;

a.x=p1.x-p0.x;

a.y=p1.y-p0.y;

b.x=p2.x-p0.x;

b.y=p2.y-p0.y;

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

if(t>0.0) return LEFT;

if(t<0.0) return RIGHT;

if((a.x\*a.x+a.y\*a.y)<(b.x\*b.x+b.y\*b.y)) return BEYOND;

if((a.x>0.0 && b.x<0.0) || (a.x<0.0 && b.x>0.0)) return BEHIND;

if((a.y>0.0 && b.y<0.0) || (a.y<0.0 && b.y>0.0)) return BEHIND;

if(p0.x==p2.x && p0.y==p2.y) return ORIGIN;

if(p1.x==p2.x && p1.y==p2.y) return DESTINATION;

return BETWEEN;

}

1. **Cohen-Sutherland Polygon Clipping**

typedef struct {

double x,y;

}Point;

enum { LEFT=0,RIGHT,BEHIND,BEYOND,ORIGIN,DESTINATION,BETWEEN};

enum { COLLINEAR=0,PARALLEL,SKEW,SKEW\_CROSS,SKEW\_NO\_CROSS};

int intersect(Point a,Point b,Point c,Point d,double \*t) {

double denom,num;

int aclass;

Point n,ba,ac;

n.x = d.y - c.y; ba.x = b.x - a.x; ac.x = a.x - c.x;

n.y = c.x - d.x; ba.y = b.y - a.y; ac.y = a.y - c.y;

denom = dotProduct(n,ba);

if(is\_equal(denom,0.0)) {

aclass = classify(c,d,a);

if(aclass == LEFT || aclass == RIGHT) return PARALLEL;

else return COLLINEAR;

}

num = dotProduct(n,ac);

\*t = -num / denom;

return SKEW; }

double dotProduct(Point a,Point b) { return a.x \* b.x + a.y \* b.y; }

intlineClip(Pointa,Pointb,Pointp[],int n) {

Point r[SIZE];

Point org,dest,crosspt;

int orgInside,destInside;

double t; int i,j;

p[n] = p[0];

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

org = p[i]; dest = p[i+1];

orgInside = (classify(a,b,org)!=LEFT);

destInside = (classify(a,b,dest)!=LEFT);

if(orgInside != destInside) {

intersect(a,b,org,dest,&t);

crosspt.x=a.x+t\*(b.x - a.x); crosspt.y=a.y + t\*(b.y-a.y); }

if(orgInside && destInside) r[j++] = dest;

else if(orgInside && !destInside) {

if((!is\_equal(org.x,crosspt.x))||(!is\_equal(org.y,crosspt.y)))

r[j++] = crosspt;

} else if(!orgInside && !destInside) ;

else { r[j++] = crosspt;

if((!is\_equal(dest.x,crosspt.x))||(!is\_equal(dest.y,crosspt.y)))

r[j++] = dest; } }

for(i=0;i<j;i++) p[i] = r[i];

return j;

}

int polygonClip(Point subject[],int m,Point clipper[],int n) {

// polygons have to be in //clockwise order

int tm,i;

clipper[n] = clipper[0];

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

tm=lineClip(clipper[i],clipper[i+1],subject,m);

m = tm; } return m; }

1. **TetraHedron Formula :**

inline double volume(double u, double v, double w, double U, double V, double W) {

    double u1,v1,w1;

    u1 = v \* v + w \* w - U \* U;

    v1 = w \* w + u \* u - V \* V;

    w1 = u \* u + v \* v - W \* W;

    return sqrt(4.0\*u\*u\*v\*v\*w\*w - u\*u\*u1\*u1 - v\*v\*v1\*v1 - w\*w\*w1\*w1 + u1\*v1\*w1) / 12.0;

}

inline double surface(double a, double b, double c) {

    return sqrt((a + b + c) \* (-a + b + c) \* (a - b + c) \* (a + b - c)) / 4.0; } inline double insphere(double WX, double WY, double WZ, double XY, double XZ, double YZ) { double sur, rad;sur = surface(WX, WY, XY) + surface(WX, XZ, WZ) + surface(WY, YZ, WZ) + surface(XY, XZ, YZ);

rad = volume(WX, WY, WZ, YZ, XZ, XY) \* 3.0 /sur;return rad; }

**Data Structure:**

1. **LCA :**

void process3(int N, int T[MAXN], int P[MAXN][LOGMAXN]) {

int i, j;

for (i = 0; i < N; i++) for (j = 0; (1 << j) < N; j++) P[i][j] = -1;

for (i = 0; i < N; i++) P[i][0] = T[i];

for (j = 1; (1 << j) < N; j++) for (i = 0; i < N; i++) if (P[i][j - 1] != -1)

P[i][j] = P[P[i][j - 1]][j - 1]; }

int query(int N, int P[MAXN][LOGMAXN], int T[MAXN], int L[MAXN], int p, int q) {

int tmp, log, i;

if (L[p] < L[q]) tmp = p, p = q, q = tmp;

for (log = 1; 1 << log <= L[p]; log++) ; log--;

for (i = log; i >= 0; i--) if (L[p] - (1 << i) >= L[q]) p = P[p][i];

if (p == q) return p;

for (i = log; i >= 0; i--) if (P[p][i] != -1 && P[p][i] != P[q][i])

p = P[p][i], q = P[q][i];

return T[p];

}

1. **Bit Binary Search in lg(n):**

/\*find f[idx] with cumiliative frequency = cumFre , initially bitMask=heighest bit of MaxVal, returns largest possible index.\*/

int findG(int cumFre){

int idx = 0;

while ((bitMask != 0) && (idx < MaxVal)){

int tIdx = idx + bitMask;

if (cumFre >= tree[tIdx]){

// if current cumulative frequency is equal to cumFre,

// we are still looking for //higher index (if exists)

idx = tIdx; cumFre -= tree[tIdx]; }

bitMask >>= 1; }

if (cumFre != 0) return -1;

else return idx; }

1. **Treap :**

struct T {

int key, priority;

T \*left, \*right;

T() {}

T(int key, int priority, T\* left, T\* right) {

this->key = key;

this->priority = priority;

this->left = left, this->right = right;

}

} \*R, \*nil; // nil indica un nod 'gol'

void init(T\* &R) {

srand(unsigned(time(0)));

R = nil = new T(0, 0, NULL, NULL);

}

int search(T\* n, int key) {

if (n == nil) return 0;

if (key == n->key) return 1;

if (key < n->key) return search(n->left, key);

else return search(n->right, key);

}

void rotleft(T\* &n) {

T \*t = n->left;

n->left = t->right, t->right = n; n = t;

}

void rotright(T\* &n) {

T \*t = n->right;

n->right = t->left, t->left = n; n = t;

}

void balance(T\* &n) {

if (n->left->priority > n->priority) rotleft(n);

else if (n->right->priority > n->priority) rotright(n);

}

// call with insert(R, key, rand() + 1);

void insert(T\* &n, int key, int priority) {

if (n == nil) {

n = new T(key, priority, nil, nil);

return;

}

if (key < n->key) insert(n->left, key, priority);

else if (key > n->key) insert(n->right, key, priority);

balance(n);

}

void erase(T\* &n, int key) {

if (n == nil) return;

if (key < n->key) erase(n->left, key);

else if (key > n->key) erase(n->right, key);

else {

if (n->left == nil && n->right == nil) delete n, n = nil;

else {

(n->left->priority > n->right->priority) ? rotleft(n) : rotright(n);

erase(n, key);

}

}

}

void split(T\* &R, T\* &Ts, T\* &Tg, int key) {

insert(R, key, infinity);

Ts = R->left, Tg = R->right;

delete R, R = nil;

}

void join(T\* &R, T\* Ts, T\* Tg, int key) {

R = new T(key, 0, Ts, Tg);

erase(R, R->key);

}

**Number Theory / Math :**

1. **Tonelli-Shank’s Modulo Square Root:**

LLD M ;

LLD legendre\_symbol(LLD a, LLD p){

LLD ls = big(a, (p - 1) / 2, p);

if (ls == p - 1) return -1 ;

else return ls ;

}

LLD FindS(LLD p){

LLD s, e; s = p - 1; e = 0;

while (s % 2 == 0){

s /= 2; e += 1; }

return s;

}

static LLD findE(LLD p) {

LLD s, e; s = p - 1; e = 0;

while (s % 2 == 0) { s /= 2; e += 1; } return e; }

static LLD Ord(LLD b, LLD p) {

LLD m = 1; LLD e = 0;

while (big(b,m,p) != 1) {

m \*= 2; m %= (M-1) ;e++; }

return e; }

static LLD TwoExp(LLD e) {

LLD a = 1;

while (e > 0) {

a \*= 2; a %= (M-1) ; e-; }

return a; }

/\*\*\*\*\*\*\* given P(P is a prime),a(0<=a<P), solves the congruence (x^2)%P = a ;

returns 0 if no solution, if returned solution is x1, then another solution is x2 = P-x1, \*\*/

static LLD ShanksSqrt(LLD a,LLD p) {

M = p ; if (legendre\_symbol(a, p) != 1) return 0 ;

else if (a == 0) return 0 ;

else if (p == 2) return p ;

if(big(a,(p - 1) /2,p)==(p- 1))

return 0; //No Sqrt Exists

if (p % 4 == 3)

return big(a,(p + 1) / 4, p);

LLD s, e; s = FindS(p); e = findE(p); LLD n, m, x, b, g, r; n = 2;

while(big(n,(p -1)/2,p)==1) n++;

x = big(a,(s + 1) / 2, p);

b = big(a,s, p);g = big(n,s, p);

r = e; m = Ord(b, p);

if (m == 0) return x;

while (m > 0) {

x = (x \* big(g,TwoExp(r - m - 1), p)) % p;

b = (b \* big(g,TwoExp(r - m), p)) % p;

g = big(g,TwoExp(r - m), p);

r = m; m = Ord(b, p); }

return x; }

1. **Shank’s Baby Step-Giant Step :**

/\* for the equation: b = a^x % p where a, b, p known, finds x works only when p > 2 \*/

int shank(int a, int b, int p) {

    int i, j, m;

    long long c, aj, ami;

    map< long long, int > M;

    map< long long, int > :: iterator it;

 m=(int)ceil(sqrt((double)(p)));

   M.insert(make\_pair(1, 0));

for(j = 1, aj = 1; j < m; j++) {

       aj = (aj \* a) % p;

    M.insert(make\_pair(aj, j));

    }

ami =modexp(modinv(a, p), m, p);

 for(c = b, i = 0; i < m; i++) {

      it = M.find(c);

        if(it != M.end()) return i \* m + it->second;

        c = (c \* ami) % p;

    }

    return 0; }

1. **FFT:**

int rev[MAXN];

base wlen\_pw[MAXN];

void fft (base a[], int n, bool invert) { // optimized

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

if (i < rev[i])

swap (a[i], a[rev[i]]);

for(int len=2; len<=n; len<<=1){

double ang = 2\*PI/len \* (invert?-1:+1);

int len2 = len>>1;

base wlen (cos(ang), sin(ang));

wlen\_pw[0] = base (1, 0);

for (int i=1; i<len2; ++i)

wlen\_pw[i] = wlen\_pw[i-1] \* wlen;

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

base t, \*pu = a+i, \*pv = a+i+len2, \*pu\_end = a+i+len2,

\*pw = wlen\_pw;

for (; pu!=pu\_end; ++pu, ++pv, ++pw) {

t = \*pv \* \*pw; \*pv = \*pu - t;

\*pu += t; }}}if (invert)

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

a[i] /= n; }

void calc\_rev (int n,int log\_n){

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

rev[i] = 0;

for (int j=0; j<log\_n; ++j)

if (i & (1<<j))

rev[i] |= 1<<(log\_n-1-j);

} }

typedef complex<double> base;

void fft (vector<base> & a, bool invert) {

int n = (int) a.size();

for (int i=1, j=0; i<n; ++i) {

int bit = n >> 1;

for (; j>=bit; bit>>=1) j -= bit; j += bit;

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

for(int len=2;len<=n; len<<=1){

double ang = 2\*PI/len \* (invert ? -1 : 1);

base wlen (cos(ang), sin(ang));

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

base w (1);

for (int j=0; j<len/2; ++j) {

base u=a[i+j],v= a[i+j+len/2]\*w;

a[i+j] = u + v;

a[i+j+len/2] = u - v; w \*= wlen;

}}} if (invert)

for(int i=0; i<n;++i)a[i]/= n;}

void multiply (const vector<int> & a, const vector<int> & b, vector<int> & res) {

vector<base> fa (a.begin(), a.end()), fb (b.begin(), b.end());

size\_t n = 1;

while (n < max (a.size(), b.size())) n <<= 1;

n <<= 1; fa.resize (n), fb.resize (n);

fft (fa, false), fft (fb, false);

for (size\_t i=0; i<n; ++i)

fa[i] \*= fb[i]; fft (fa, true);

res.resize (n);

for (size\_t i=0; i<n; ++i)

res[i] = int(fa[i].real()+ 0.5);

}

1. **minimum number of knight moves in an infinite chess board to go from position (x1,y1) to position (x2,y2):**

int64 dist(int64 x1, int64 y1, int64 x2, int64 y2){

int64 dx = abs(x2-x1);

int64 dy = abs(y2-y1);

int64 lb=(dx+1)/2;

lb = max(lb, (dy+1)/2);

lb = max(lb, (dx+dy+2)/3);

while ((lb%2)!=(dx+dy)%2) lb++;

if(abs(dx)==1 &&dy==0)return 3;

if (abs(dy)==1&&dx==0)return 3;

if (abs(dx)==2 && abs(dy)==2)return 4;return lb;}

1. **Pollard-Rho:**

public static BigInteger rho(BigInteger N) {

BigInteger divisor;

BigInteger c = new BigInteger(N.bitLength(), random);

BigInteger x = new BigInteger(N.bitLength(), random);

BigInteger xx = x;

if (N.mod(TWO).compareTo(ZERO) == 0) return TWO;

do {

x=x.multiply(x).mod(N).add(c).mod(N);

xx=xx.multiply(xx).mod(N).add(c).mod(N);

xx=xx.multiply(xx).mod(N).add(c).mod(N);

divisor = x.subtract(xx).gcd(N); } while((divisor.compareTo(ONE)) == 0); return divisor; }

1. **Determinant Without Double (Matrix Tree Theorem) :**

int N,a; LL m[40][40];

LL determinant() {LL coeff=1,min,Sum,index,row=0,col=0,tmp;

while((row!=N-1)&&(col!=N-1)){

FOR(j,col,N) if(m[row][j] < 0){

FOR(i,row,N) m[i][j] =-m[i][j];

coeff = -coeff; }

if(!m[row][col]) FOR(j,col+1,N) if(m[row][j]) {

FOR(i,row,N)swap(m[i][col],m[i][j]);

coeff = -coeff; break; }

if(!m[row][col]) return 0; Sum=0;

FOR(j,col+1,N) Sum += m[row][j];

while(Sum) {

min=m[row][col]; index=col;

FOR(j,col+1,N)if((min>m[row][j])&&(m[row][j]>0)) {

index=j; min=m[row][j]; }

if(index>col) { FOR(i,row,N) swap(m[i][col],m[i][index]);

coeff=-coeff; }

FOR(j,col+1,N) if(m[row][col]){

tmp=m[row][j]/m[row][col];

FOR(i,row,N) m[i][j] -= tmp\*m[i][col]; } Sum=0;

FOR(j,col+1,N)Sum += m[row][j];}

coeff\*=m[row][col];row++; col++;

} return (coeff\*m[row][col]); }

int main(){

scanf("%d%d",&N,&a);

REP(i,N) REP(j,N) m[i][j] = 0;

REP(i,a){int p,q; scanf("%d%d",&p,&q); p--; q--;m[p][q] = m[q][p] = -1; }

REP(i,N) REP(j,N) if(j!=i) m[i][i] -= m[i][j];

if(N == 1){ puts("1");continue;}

N--; cout<<determinant()<<endl;

return 0; }

1. **Extended-Euclid :**

int x,y,d; //ax+by=d=gcd(a,b)

void extended\_euclid(int a,int b)/\*minimizes |x|+|y| (primarily), then x<=y (secondarily).\*/{

int r,q,x0,y0,x1,y1;

x=y1=0; x1=y=1; r=a%b;

while(r) {

x0=x1; x1=x; y0=y1; y1=y;

q=a/b; a=b; b=r; x=x0-x1\*q;

y=y0-y1\*q; r=a%b; } d = b; }

1. **Right-Shift An Array By M bits :**

void rshift(int m){

int pos1 = 0, bit1 = 0, i ;

int pos2 = m>>6, bit2 = m&63 ;

rep(i,0,pos2+1) tmp[i] = dp[i] ;

while(pos2<mx){

if(bit1>bit2) { dp[pos2] |= ((LLU)(tmp[pos1]>>bit1)<<bit2) ;

bit2 += (64-bit1) ; pos1 ++ ;

bit1 = 0 ; }else if(bit1<bit2) {

dp[pos2] |= ((LLU)((tmp[pos1]>>bit1)&pre[bit2])<<bit2) ; bit1 += (64-bit2) ; pos2 ++ ; bit2 = 0 ;tmp[pos2] = dp[pos2] ;

} else { dp[pos2] |= ((LLU)(tmp[pos1]>>bit1)<<bit2) ;

pos1++;pos2++;bit1 = bit2 = 0 ; tmp[pos2] = dp[pos2] ; }}}

# Construct n from the Sum of Its Divisors:

/\* powi64(a, b) computes a^b, rememver that prime upto i-1 are used\*/

i64 table[NN+1][NN+1]; /\* if there is an overflow, table[i][j] = inf;\*/

void preprocessTable() {

for( int i = 0; i <= NN; i++ ) table[0][i] = 1;

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

table[i][0] = 1;

for( int j = 1; j < NN; j++ ) table[i][j] = table[i][j-1] + powi64(pr[i-1], j); } }

vector <i64> calculateXFromSumOfDivisors( int sum ) {

vector <i64> res;i64 val = 1, prevD = 1;

for( int i = NN; ; i-- ) {

if( sum == 1 ) {

res.push\_back( val ); /\* Here the value is saved\*/

sum \*= prevD, val = 1; }

if( i <= 0 || sum == 1 ) break;

for( int j =NN - 1;j >= 0; j--){

if( table[i][j] > 1 && ( sum % table[i][j] == 0 ) ) {

val \*= powi64( pr[i-1], j );

sum /= table[i][j], prevD = table[i][j];

break; }}} return res; }

1. **All Real Roots of An N Degree Equation :**

#define INF 1000000.

#define myabs(x) ((x>0)?(x):-(x))

double f(intn,doublex,double\*a){

inti;double s=0;

for(i=0; i<=n; i++) s+=a[i]\*pow(x,i); return s; }

double bisect(int n,double \*a,double lm,double um,int fx){ double x,y,LMT = 1e-13; while(1){ x=(um+lm)/2.; y=f(n,x,a); if(myabs(y) <LMT || um-lm < LMT) break; else {

if(fx==1) {

if(y > LMT) um=x; else lm=x; }

else { if(y > LMT) lm=x; else um=x; }}} return x; }

void Differentiate(int n,double \*a,double \*b){ int i;

// d/dx(f(n,x,a)) = f(n-1,x,b)

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

b[i]=(i+1)\*a[i+1];if(b[n] < 0)

for(i=0; i<=n; i++) b[i]= -b[i];

} int main(){ int i,j,k,n,r[31];

doublea[30][31],x[30][31],y[31],D;

while(scanf("%d",&n)==1 && n){

for(i=n; i>=0; i--)

scanf("%lf",&a[n][i]);if(n==1) {

printf("Equation of Degree 1\n");

printf("Real Roots(1), Complex Roots(0)\n"); printf("%.10lf\n",-a[1][0] / a[1][1]); continue; } for(i=n-1; i>=2; i--) Differentiate(i,a[i+1],a[i]);

D=a[2][1]\*a[2][1]- 4\*a[2][2]\*a[2][0];

if(D < 0) {

x[2][0]=-INF;x[2][1]=INF;r[2]=2;

} else {

x[2][0]=-INF;

x[2][1]=(-a[2][1]-sqrt(D))/(2\*a[2][2]);

x[2][2]=(-a[2][1]+sqrt(D))/(2\*a[2][2]);

x[2][3]=INF; r[2]=4; }

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

for(j=0; j < r[i]; j++)

y[j]=f(i+1,x[i][j],a[i+1]);

k=0; x[i+1][k++]=-INF;

for(j=0; j < r[i]-1; j++){

if( y[j] >=0 && y[j+1] <=0)

x[i+1][k++] = bisect( i+1 , a[i+1] , x[i][j] , x[i][j+1] , -1);

else if( y[j] <=0 && y[j+1] >=0)

x[i+1][k++] = bisect( i+1 , a[i+1] , x[i][j] , x[i][j+1] , 1); }

x[i+1][k++]=INF; r[i+1]=k; }

printf("Equation of Degree %d\n",n);

printf("Real Roots(%d), Complex Roots(%d)\n",r[n]-2,n-r[n]+2);

for(i=1; i<r[n]-1; i++) {

if(i>1) printf(" ");

printf("%.10lf",x[n][i]); }

printf("\n"); } return0; }

1. **Josephus :**

int f(int n){// 1 based. for k=1

if(n == 1) return 1;return (f((n-(n&1))>>1)<<1) + ((n&1)?1:-1); }

int f(int n, int k) { // o based

if(n == 1) return 0;

  return (f(n-1, k) + k)%n; }

1. **Bitwise & Segmented Seive :**

#define MAX 100100

#define LMT 400 //sqrt(MAX)

#define chkC(a,n) (a[n>>6]&(1<<((n>>1)&31)))

#define setC(a,n) (a[n>>6]|=(1<<((n>>1)&31)))

int flag[MAX/64];

int segment[MAX/64];

unsigned primes[10000];

int prlen ; void sieve(){

unsigned i,j,k;flag[0]|=0;

for(i=3;i<LMT;i+=2)if(!chkC(flag,i))for(j=i\*i,k=i<<1;j<MAX;j+=k)setC(flag,j); j=0; primes[j++] = 2; for(i=3;i<MAX;i+=2)if(!chkC(flag,i))primes[j++] = i;prlen = j ;}

int segSieve(int a, int b){

int rt, i, k, cnt = (a<=2 && 2<=b)? 1 : 0;unsigned j ; if(b<2) return 0;

if(a<3) a = 3; if(a%2==0) a++;

CLR(segment) ;

for(i=1,rt=(int)sqrt((double)b); i < prlen && primes[i] <= rt; i++){j = primes[i] \* ( (a+primes[i]-1) / primes[i] );if(j%2==0) j += primes[i];for(k=primes[i]<<1; j<=b; j+=k) if(j!=primes[i]) setC(segment, (j-a));}for(i=0; i<=b-a; i+=2) if(!chkC(segment, i)) cnt++;return cnt;}

**13.Gaussian Elimination:**

void gaussian(int r, int c){

int i=0, j = 0,k,l ; LF val ; while(i<r && j<c) { rep(k,i,r) { if(!(fabs(a[k][j])<eps)) { rep(l,j,c) swap(a[i][l],a[k][l]) ; swap(b[i],b[k]) ; break ;}} if(k==r) { j++ ; continue ;} rep(k,i+1,r) { val = a[k][j]/a[i][j] ; rep(l,j,c) { a[k][l] -= a[i][l]\*val ; }

b[k] -= b[i]\*val ; } i++;j++ ;} /\*rep(k,i,r) {

rep(j,0,c) if(!(fabs(a[k][j])<eps)) goto stop ; if(!(fabs(b[k])<eps)) return -1 ; // no solution stop : ; }

if(i>c)return -1 ;// no solution if(i==c)return 0;//unique //solution

if(i<c) return 1 ; // multiple //solution \*/

rem(i,c-1,-1) {

x[i] = b[i] ;

rep(k,i+1,c) x[i] -= a[i][k]\*x[k] ;

x[i] /= a[i][i] ; } }

**Cheat Sheet :**

**Geometric Formula:**

|  |  |
| --- | --- |
| **Triangle** | Circum Radius = a\*b\*c/(4\*area)  In Radius = area/s, where s = (a+b+c)/2  length of median to side c = sqrt(2\*(a\*a+b\*b)-c\*c)/2  length of bisector of angle C = sqrt(ab[(a+b)\*(a+b)-c\*c])/(a+b) |
| **Ellipse** | Area = PI\*a\*b  Circumference = 4a \*int(0,PI/2){sqrt(1-(k\*sint)\*(k\*sint))}dt            = 2\*PI\*sqrt((a\*a+b\*b)/2) approx         where k = sqrt((a\*a-b\*b)/a)            = PI\*(3\*(r1+r2)-sqrt[(r1+3\*r2)\*(3\*r1+r2)]) |
| **Spherical cap** | V = (1/3)\*PI\*h\*h\*(3\*r-h)  Surface Area = 2\*PI\*r\*h |
| **Spherical Sector** | V = (2/3)\*PI\*r\*r\*h |
| **Spherical Segment** | V=(1/6)\*PI\*h\*(3\*a\*a+3\*b\*b+h\*h) |
| **Torus** | V = 2\*PI\*PI\*R\*r\*r |
| **Truncated Conic** | V = (1/3)\*PI\*h\*(a\*a+a\*b+b\*b)  Surface Area = PI\*(a+b)\*sqrt(h\*h+(b-a)\*(b-a))           = PI\*(a+b)\*l |
| **Pyramidal frustum** | (1/3)\*h\*(A1+A2+sqrt(A1\*A2)) |

**Trigonometric Functions:**

tan A/2 = +sqrt((1-cos A)/(1+cos A))= sin A / (1+cos A)

    = (1-cos A) / sin A

    = cosec A – cot A

sin 3A  = 3\*sin A–4\*sincube A cos 3A  = 4\*coscube A – 3\*cos A

tan 3A  = (3\*tan A-tancube A)/(1-3\*tansq A)

sin 4A  = 4\*sin A\*cos A – 8\*sincube A\*cos A

cos 4A  = 8\*cos4 A – 8\*cossq A + 1

[r\*(cost+i\*sint)]p = rp\*(cos pt+i\*sin pt)

**a**cos**x** + **b**sin**x = c,** x = 2n +  ± , where

cos = a / (sqrt(a^2+b^2)), cos = c / (sqrt(a^2+b^2));

**Integration Formula:**

a^x => a^x/ln(a)

1/sqrt(x\*x+a\*a) => ln(x+sqrt(x\*x+a\*a))

1/sqrt(x\*x-a\*a) => ln(x+sqrt(x\*x-a\*a))

1/(x\*sqrt(x\*x+a\*a) => -(1/a)\*ln([a+sqrt(x\*x+a\*a)]/x)

1/(x\*sqrt(a\*a-x\*x) => -(1/a)\*ln([a+sqrt(a\*a-x\*x)]/x)

**Differentiation Formula:**

asin x => 1/sqrt(1-x\*x)                    acos x => -1/sqrt(1-x\*x) atan x => 1/(1+x\*x)                         acot x => -1/(1+x\*x)

asec x => 1/[x\*sqrt(x\*x-1)]

acosec x => -1/[x\*sqrt(x\*x-1)]

a^x =>a^x\*ln(x)

cot x => -cosecsq x

sec x =>sec x\* tanx

cosec x => -cosec x \* cot x

**Rotation Matrice :**

Qx(T) = (1,0,0)(0,cosT,-sinT)(0,sinT,cosT)

Qy(T) = (cosT,0,sinT)(0,1,0)(-sinT,0,cosT)

 Qz(T) = (cosT,-sinT,0)(sinT,cosT,0)(0,0,1)

Q2x2(T) = (cosT,-sinT)(sinT,cosT)

**GreatCircleDist**(lat1,long1,lat2,long2,r) =(r\*acos(sin(lat1)\*sin(lat2)+cos(lat1)\*cos(lat2)\*cos(long1-long2)))

**picks’s theorem:** I = area + 1 - B/2, I = number of points inside, B = number Of points on the border

**Misere Nim:** s[i]>=1, if(countOne(stone[])==n-1)  ans = 1; else if(countOne(stone[])==n) ans = !(n&1) ; else ans = xor(all s[])

**Birthday Paradox:** P(n)~1-e^(-n\*n/(2\*d))=1-npr(d,n)/d^n, num of people necessary to have >=50% chance of matching n~.5+sqrt(.25-2\*d\*ln(.5))

**modular arithmetic:**if(x≡y(mod phi(n))) then a^x≡a^y(mod n), gcd(a,n)=1

**number of dag’s on n nodes:** dp(n)=for(k=1;k<=n;k++) (-1)^(k-1)\*ncr(n,k)\*2^(k\*(n-k))\*dp(n-k)

**xor of all num [0,a]:** ret[]={a,1,a+1,0}; return ret[a%4];

**summation of power:** [1<=i<=n], sum(i^m) = [ (n+1)m+1-1-sum( (i+1)m+1-im+1-(m+1)\*im ) ] / (m+1) ;

**Newton’s Forward difference Interpolation Formula:**  n,x[],y[],given. xp=x0+p\*h, p=[0,n-1], the n degree polynomial is,

f(p)= [0<=k<=n] sum( ncr(p,k)∆ky0 )

**N-puzzle solvability:** ( (grid width odd) && (#inversions even) )  ||  ( (grid width even) && ((blank on odd row from bottom) == (#inversions even)) )

**Derangement:** D(n)=(n-1)\*(D(n-1)+D(n-2));

**Burnside’s Lemma:**1 cubes, n colors,num of rotationally distinct coloring is, (n6+3n4+12n3+8n2)/24

**Fibonacci:** gcd(Fn,Fm) = Fgcd(n,m) ;  F(n) =  ( a^n - (-1/a)^n ) / sqrt(5) = ( a^n - (1-a)^n ) / sqrt(5) ; a = (1+sqrt(5))/2; -1/a=(1-a) ;

**Check if AxB==C:**3 nxn matrices A,B,C,generate nx1 rand mat r, p=Ax(Bxr)-(Cxr), if p==(0..0) then yes.run >=20 times.

**Chinese Remainder Theorem :** x≡r1(mod m1), x≡r2(mod m2),… x≡rk(mod mk), mi are pairwise coprime, the system has a unique solution modulo M=m1m2…mk . let Mi=M/mi ,let si be the solution of Mix≡1(mod mi) … then X0=(M1s1r1+M2s2r2… Mkskrk)%M is a solution of the system .