**Menu:**

template.........................................02

Articulation Point...............................03

Bellman Ford.....................................03

Floyd Warshall...................................04

MaxFlow (Dinitz).................................04

MinCost MaxFlow(Dijkstra)........................06

MinCost MaxFlow(Bellman Ford)....................08

Matching(bpm)....................................10

Matching(hopcroft karp)..........................11

Strongly Connected Component(+component graph)...12

2-SAT............................................14

Centroid Decomposition...........................17

Dominator Tree...................................19

KMP..............................................20

Z-algorithm......................................21

Suffix Array.....................................22

Manachar.........................................25

Palindromic Tree.................................25

LIS NlongN.......................................27

ConvexHull Trick.................................27

Divide and Conquer Trick.........................29

Extended Euclid..................................30

Gauss............................................33

CRT..............................................35

Binary Indexed Tree..............................36

Disjoint Set Union...............................36

Heavy Light Decomposition........................37

LCA..............................................38

SQRT Decomposition...............................39

Mo's Algorithm...................................40

Treap............................................42

Implicit Treap...................................43

Postfix and Result...............................45

String X String..................................47

Shank's baby-step giant-step algorithm...........49

Stable Marriage Problem..........................50

Convex Hull(Minimized)...........................50

Convex Hull(Maximized)...........................51

Convex Hull Diameter.............................53

Closest Pair.....................................54

Utilities........................................55

Directed MST.....................................62

Simplex..........................................64

Egyptian Factorization...........................67

Hashing..........................................68

2-Edge Split.....................................70

Russian Peasant..................................71

**Template:**

#include <bits/stdc++.h>

#define rep(i,n) for(i=1;i<=n;i++)

#define Rep(i,n) for(i=0;i<n;i++)

#define For(i,a,b) for(i=a;i<=b;i++)

#define pb(x) push\_back(x)

#define sz(x) x.size()

#define mem(ara,val) memset(ara,val,sizeof(ara))

#define eps 1e-9

#define si(x) scanf("%d",&x)

#define sii(x,y) scanf("%d %d",&x,&y)

#define siii(x,y,z) scanf("%d %d %d",&x,&y,&z)

#define sl(x) scanf("%lld",&x)

#define sll(x,y) scanf("%lld %lld",&x,&y)

#define slll(x,y,z) scanf("%lld %lld %lld",&x,&y,&z)

#define ss(ch) scanf("%s",ch)

#define pi(x) printf("%d",x)

#define pii(x,y) printf("%d %d",x,y)

#define piii(x,y,z) printf("%d %d %d",x,y,z)

#define pl(x) printf("%lld",x)

#define pll(x,y) printf("%lld %lld",x,y)

#define plll(x,y,z) printf("%lld %lld %lld",x,y,z)

#define ps(ch) printf("%s",ch)

#define Afridi 0

#define NL printf("\n")

#define SP printf(" ")

#define debug(x) cout << #x << " " << x << endl

#define Max 1000005

#define INF INT\_MAX

#define mod 1000000007

#define FI freopen("in.txt","r",stdin)

#define FO freopen("out.txt","w",stdout)

#define D(x) cout << #x << " = " << x << endl

#define DD(x,y) cout << #x << " = " << x << " " << #y << " = " << y << endl

typedef long long LL;

typedef unsigned long long ULL;

using namespace std;

LL bigmod(LL b,LL p)

{

if(p == 0)return 1;

LL my = bigmod(b,p/2);

my\*=my; my%=mod;

if(p & 1)my\*=b,my%=mod;

return my;

}

int setb(int n,int pos){return n=n | (1 << pos);}

int resb(int n,int pos){return n=n & ~(1 << pos);}

bool checkb(int n,int pos){return (bool)(n & (1 << pos));}

**Graph Theory:**

**Articulation Point:**

void AP(LL u)

{

dis[u] = low[u] = ++tm;

LL child = 0;

visit[u] = 1;

LL len = g[u].size();

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

{

v = g[u][i];

if(visit[v] == 0)

{

child++;

par[v] = u;

AP(v);

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

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

if(par[u] != -1 && low[v] >= dis[u])ap[u] = 1;

if(low[v] > dis[u])bridge;

}

else if(v != par[u])

{

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

}

}

}

**Bellman Ford:**

void bellman()

{

LL i,j;

for(i=0; i,v; i++)dis[i] = INF;

dis[s] = 0;

for(i=1; i<=v-1; i++)

{

for(j=0; j<e; j++)

{

LL u = ed[j].u;

LL v = ed[j].v;

LL w = ed[j].w;

if(dis[u] != INF && dis[u]+w < dis[v])

{

dis[v] = dis[u] + w;

}

}

}

for(j=0; j<e; j++)

{

LL u = ed[j].u;

LL v = ed[j].v;

LL w = ed[j].w;

if(dis[u] != INF && dis[u]+w < dis[v])

{

///cycle detected;

}

}

}

**Floyd Warshall:**

void floyed()

{

LL dis[n][n],i,j,k;

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

{

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

}

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

{

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

{

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

{

if(dist[i][k] + dist[k][j] < dist[i][j])

{

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

}

}

}

}

}

par[i][j] = par[j][k];

**MaxFlow (Dinitz):**

#define SET(p) memset(p, -1, sizeof(p))

#define CLR(p) memset(p, 0, sizeof(p))

#define i64 long long

const int MAXN = 20010, MAXE = 180006;

int src, snk, nNode, nEdge;

int Q[MAXN], fin[MAXN], pro[MAXN], dist[MAXN];

int flow[MAXE], cap[MAXE], nxt[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 c)

{

to[nEdge] = v, cap[nEdge] = c, flow[nEdge] = 0, nxt[nEdge] = fin[u], fin[u] = nEdge++;

to[nEdge] = u, cap[nEdge] = 0, flow[nEdge] = 0, nxt[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=nxt[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=nxt[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;

}

int dinitz()

{

int 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 += df;

else break;

}

}

return ret;

}

**MinCost MaxFlow(Dijkstra):**

#define Pot(u,v) (d[u] + pi[u] - pi[v])

LL cap[Max][Max],cost[Max][Max];

LL fnet[Max][Max],adj[Max][Max], deg[Max];

LL par[Max], d[Max];

LL pi[Max];

LL n,m,s,t,fcost;

bool dijkstra()

{

LL i;

for(i = 0; i < n; i++ ) d[i] = INF, par[i] = -1;

d[s] = 0;

par[s] = -n - 1;

while( 1 )

{

LL u = -1, bestD = INF;

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

{

if( par[i] < 0 && d[i] < bestD )

{

bestD = d[u = i];

}

}

if( bestD == INF ) break;

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

for(i = 0; i < deg[u]; i++ )

{

LL v = adj[u][i];

if( par[v] >= 0 ) continue;

if( fnet[v][u] && d[v] > Pot(u,v) - cost[v][u] )

{

d[v] = Pot( u, v ) - cost[v][u], par[v] = -u-1;

}

if( fnet[u][v] < cap[u][v] && d[v] > Pot(u,v) + cost[u][v] )

{

d[v] = Pot(u,v) + cost[u][v], par[v] = -u - 1;

}

}

}

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

{

if( pi[i] < INF )

{

pi[i] += d[i];

}

}

return par[t] >= 0;

}

LL mcmf3()

{

mem(deg,0);

LL i,j;

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

{

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

{

if( cap[i][j] || cap[j][i] )

{

adj[i][deg[i]++] = j;

}

}

}

mem(fnet,0);

mem(pi,0);

LL flow = 0;

fcost = 0;

while( dijkstra() )

{

LL bot = INF;

for( LL v = t, u = par[v]; v != s; u = par[v = u] )

{

bot = min(bot,fnet[v][u] ? fnet[v][u] : ( cap[u][v] - fnet[u][v] ));

}

for( LL v = t, u = par[v]; v != s; u = par[v = u] )

{

if( fnet[v][u] )

{

fnet[v][u] -= bot;

fcost -= bot \* cost[v][u];

}

else

{

fnet[u][v] += bot;

fcost += bot \* cost[u][v];

}

}

flow += bot;

}

return flow;

}

///----------------- EXAMPLE USAGE -----------------

int main()

{

cin >> n;

mem(cap,0);mem(cost,0);

int m, a, b, c, cp;

cin >> m;

cin >> s >> t;

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

{

cin >> a >> b >> cp >> c;

cost[a][b] = c; /// cost[b][a] = c;

cap[a][b] = cp; /// cap[b][a] = cp;

}

int flow = mcmf3();

cout << "flow: " << flow << endl;

cout << "cost: " << fcost << endl;

return 0;

}

**MinCost MaxFlow(Bellman Ford):**

#define SET(p) memset(p, -1, sizeof(p))

#define CLR(p) memset(p, 0, sizeof(p))

#define i64 long long

#define MAXN 125

#define MAXE 30000

LL src, snk, nNode, nEdge;

LL fin[MAXN], pre[MAXN], dist[MAXN];

LL cap[MAXE], cost[MAXE], nxt[MAXE], to[MAXE], from[MAXE];

inline void init(LL \_src, LL \_snk, LL nodes)

{

SET(fin);

nNode = nodes, nEdge = 0;

src = \_src, snk = \_snk;

}

inline void add(LL u, LL v, LL \_cap, LL \_cost)

{

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

nxt[nEdge] = fin[u], fin[u] = nEdge++;

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

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

}

bool bellman()

{

LL 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 = nxt[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);

}

LL mcmf(LL &fcost)

{

LL 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;

}

**Matching(bpm):**

LL Left[Max],Right[Max];

vector <LL> g[Max];

LL n,m;

bool visit[Max];

bool dfs(LL u)

{

if(visit[u] == 1)return false;

visit[u] = 1;

LL len = sz(g[u]),i;

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

{

LL v = g[u][i];

if(Right[v] == -1)

{

Left[u] = v;

Right[v] = u;

return true;

}

}

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

{

LL v = g[u][i];

if( dfs(Right[v]) )

{

Left[u] = v;

Right[v] = u;

return true;

}

}

return false;

}

LL bpm()

{

mem(Left,-1);

mem(Right,-1);

LL i;

while(1)

{

bool f = 1;

mem(visit,0);

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

{

if(Left[i] == -1 && dfs(i))

{

f = 0;

}

}

if(f)break;

}

LL cnt = 0;

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

{

if(Left[i] != -1)cnt++;

}

return cnt;

}

**Matching(hopcroft karp):**

#define MAX 80005

#define NIL 0

#define INF (1<<28)

vector< int > G[MAX];

int n, m, match[MAX], dist[MAX];

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;

mem(dist,0);

mem(match,0);

while( bfs() )

{

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

{

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

}

}

return matching;

}

**Strongly Connected Component(+component graph):**

vector <LL> g[Max],ug[Max],sc[Max],cg[Max];

bool visit[Max];

stack <LL> st;

LL idx,n,m,codx[Max];

void dfs(LL u,LL f)

{

visit[u] = 1;

LL len,i;

if(!f)len = sz(g[u]);

else len = sz(ug[u]);

Rep(i,len)

{

LL v;

if(!f)v = g[u][i];

else v = ug[u][i];

if(visit[v] == 0)dfs(v,f);

}

if(!f)st.push(u);

else

{

sc[idx].pb(u);

codx[u] = idx;

}

}

void go(LL u,LL c)

{

LL len = sz(g[u]),i;

Rep(i,len)

{

LL v = g[u][i];

if(codx[v] != c)cg[c].pb(codx[v]);

}

}

void buildGraph()

{

mem(visit,0);

LL i,j;

rep(i,idx)

{

LL len = sz(sc[i]);

if(len)sort(sc[i].begin(),sc[i].end());

Rep(j,len) {

LL u = sc[i][j];

go(u,i);

}

}

///compress

rep(i,idx)if(sz(cg[i]) > 0)sort(cg[i].begin(),cg[i].end());

rep(i,idx)

{

vector <LL> tmp;

tmp.clear();

LL prev = -1;

LL len = sz(cg[i]);

Rep(j,len)

{

LL u = cg[i][j];

if(u != prev)tmp.pb(u);

prev = u;

}

len = sz(tmp);

cg[i].clear();

Rep(j,len)cg[i].pb(tmp[j]);

}

}

int main()

{

LL i,x,y;

sll(n,m);

rep(i,m)

{

sll(x,y);

g[x].pb(y);

ug[y].pb(x);

}

mem(visit,0);

rep(i,n)if(visit[i] == 0)dfs(i,0);

mem(visit,0);

idx = 0;

while(st.empty() == 0)

{

LL u = st.top();

st.pop();

if(visit[u] == 0)

{

idx++;

dfs(u,1);

}

}

buildGraph();

rep(i,idx)

{

LL len = sz(sc[i]),j;

Rep(j,len)printf("%lld ",sc[i][j]);

NL;

}

rep(i,idx)

{

LL len = sz(cg[i]),j;

Rep(j,len)printf("%lld -> %lld\n",i,cg[i][j]);

}

return 0;

}

**2-SAT:**

vector <LL> g[Max],ug[Max],sc[Max],cg[Max];

bool visit[Max];

stack <LL> st;

LL idx,n,m,codx[Max];

void dfs(LL u,LL f)

{

visit[u] = 1;

LL len,i;

if(!f)len = sz(g[u]);

else len = sz(ug[u]);

Rep(i,len)

{

LL v;

if(!f)v = g[u][i];

else v = ug[u][i];

if(visit[v] == 0)dfs(v,f);

}

if(!f)st.push(u);

else

{

sc[idx].pb(u);

codx[u] = idx;

}

}

void go(LL u,LL c)

{

LL len = sz(g[u]),i;

Rep(i,len)

{

LL v = g[u][i];

if(codx[v] != c)cg[c].pb(codx[v]);

}

}

void buildGraph()

{

mem(visit,0);

LL i,j;

rep(i,idx)

{

LL len = sz(sc[i]);

if(len)sort(sc[i].begin(),sc[i].end());

Rep(j,len)

{

LL u = sc[i][j];

go(u,i);

}

}

///compress

rep(i,idx)if(sz(cg[i]) > 0)sort(cg[i].begin(),cg[i].end());

rep(i,idx)

{

vector <LL> tmp;

tmp.clear();

LL prev = -1;

LL len = sz(cg[i]);

Rep(j,len)

{

LL u = cg[i][j];

if(u != prev)tmp.pb(u);

prev = u;

}

len = sz(tmp);

cg[i].clear();

Rep(j,len)cg[i].pb(tmp[j]);

}

}

bool ret;

LL slol[Max];

void cdfs(LL u)

{

visit[u] = 1;

LL len = sz(cg[u]),i;

Rep(i,len)

{

LL v = cg[u][i];

if(visit[v] == 0)cdfs(v);

}

st.push(u);

}

void iiSAT()

{

LL i,j;

rep(i,n)

{

if(codx[2\*i] == codx[2\*i-1])

{

ret = 0;

return;

}

}

ret = 1;

mem(visit,0);

rep(i,idx)if(visit[i] == 0)cdfs(i);

mem(slol,-1);

vector <LL> v;

v.clear();

while(st.empty() == 0)

{

v.pb(st.top());

st.pop();

}

LL vlen = sz(v);

for(i=vlen-1; i>=0; i--)

{

LL u = v[i];

LL len = sz(sc[u]);

LL my = 1;

Rep(j,len)

{

LL v = sc[u][j];

LL d = (v+1)/2;

if(slol[d] == -1)continue;

if(v & 1)

{

if(slol[d] == 0)

{

my = 0;

break;

}

}

else

{

if(slol[d] == 1)

{

my = 0;

break;

}

}

}

Rep(j,len)

{

LL v = sc[u][j];

LL d = (v+1)/2;

if(v & 1)slol[d] = my;

else slol[d] = 1 - my;

}

}

}

int main()

{

/// (A or B)

/// !A -> B

/// !B -> A

/// SCC

/// compress

/// call iiSAT

/// if(A and !A == same)ret = 0;

/// else slol gives the solution

}

**Centroid Decomposition:**

vector <int> g[Max],ctree[Max],cost[Max]; ///tree,ctree,cost

int visit[Max],cs; ///too much visit

int centroid,s,n,croot; ///centroid = global centroid return; s = size of subtree; croot = centroid tree root

bool iscentroid[Max];

int cpar[Max],clevel[Max]; ///centroid parent and centroid level

int ctable[Max][20]; ///centroid table

int dfs(int u) ///find a centroid in a given subtree

{

visit[u] = cs;

int len = sz(g[u]),i,ret = 1,mx = 0;

Rep(i,len)

{

int v = g[u][i];

if( visit[v] != cs && iscentroid[v] == 0 )

{

int my = dfs(v);

mx = max( mx,my );

ret += my;

}

}

mx = max( mx,s-ret );

if(mx \* 2 <= s)centroid = u;

return ret;

}

int get\_size(int u) ///size of agiven subtree

{

visit[u] = cs;

int len = sz(g[u]),i,ret = 1;

Rep(i,len)

{

int v = g[u][i];

if( visit[v] != cs && iscentroid[v] == 0 )

{

ret += get\_size(v);

}

}

return ret;

}

void F(int u,int p) ///centroid decomposition(current\_node,parent)

{

cs++;s = get\_size(u);

cs++;dfs(u);

int cur = centroid;

iscentroid[cur] = 1;

if(p)

{

cpar[cur] = p;

ctree[p].pb(cur); ///build centroid tree

clevel[cur] = clevel[p] + 1;

}

else

{

croot = cur;

clevel[cur] = 0;

}

int len = sz(g[cur]),i;

Rep(i,len)

{

int v = g[cur][i];

if( iscentroid[v] == 0 )F(v,cur);

}

}

void compute\_table(int u,int d,int p)

{

visit[u] = cs;

ctable[u][ clevel[u]-clevel[p] ] = d;

int len = sz(g[u]),i;

Rep(i,len)

{

int v = g[u][i];

if( visit[v] != cs && clevel[v] > clevel[p] )

{

compute\_table(v,d+1,p);

///compute\_table(v,d+cost[u][i],p);

}

}

}

**Dominator Tree:**

vector <int> g[Max],tree[Max],bucket[Max],rg[Max];

int sdom[Max],dom[Max],par[Max],dsu[Max],label[Max],arr[Max],dep[Max],rev[Max],T;

int ed[Max][2],ok[Max];

int n,m;

int Find(int u,int x=0)

{

if(u == dsu[u])return x ? -1 : u;

int v = Find(dsu[u],x+1);

if(v < 0)return u;

if(sdom[ label[ dsu[u] ] ] < sdom[ label[u] ]) label[u] = label[ dsu[u] ];

dsu[u] = v;

return x ? v : label[u];

}

void Union(int u,int v)

{

dsu[v] = u;

}

void dfs0(int u)

{

++T; arr[u] = T; rev[T] = u;

sdom[T] = T; label[T] = T; dsu[T] = T;

for(int i = 0; i < sz(g[u]); i++)

{

int w = g[u][i];

if( !arr[w] )dfs0(w), par[ arr[w] ] = arr[u];

rg[ arr[w] ].pb( arr[u] );

}

}

void dominator\_tree\_init()

{

dfs0(1);

for(int i = T; i >= 1; i--)

{

for(int j = 0; j<sz(rg[i]); j++)

sdom[i] = min(sdom[i],sdom[ Find(rg[i][j]) ]);

if(i > 1)bucket[ sdom[i] ].pb(i);

for(int j = 0; j < sz(bucket[i]); j++)

{

int w = bucket[i][j];

int v = Find(w);

if( sdom[w] == sdom[v] )dom[w] = sdom[w];

else dom[w] = v;

}

if(i > 1)Union(par[i],i);

}

for(int i = 2; i <= T; i++)

{

if(dom[i] != sdom[i])dom[i] = dom[ dom[i] ];

tree[ rev[ dom[i] ] ].pb(rev[i]);

tree[ rev[i] ].pb(rev[ dom[i] ]);

}

}

void Clear()

{

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

{

arr[i] = dep[i] = sdom[i] = dom[i] = par[i] = dsu[i] = label[i] = rev[i] = 0;

tree[i].clear();

g[i].clear();

rg[i].clear();

bucket[i].clear();

}

T=0;

}

**String:**

**KMP:**

char text[Max],patt[Max];

LL lps[Max],ans[Max],ind;

void prework()

{

lps[0] = 0;

LL len = 0;

LL i = 1;

LL l = strlen(patt);

while(i < l)

{

if(patt[i] == patt[len])

{

len++;

lps[i] = len;

i++;

}

else

{

if(len == 0)

{

lps[i] = 0;

i++;

}

else

{

len = lps[len-1];

}

}

}

}

void kmp()

{

LL i = 0,j = 0;

LL l1 = strlen(text);

LL l2 = strlen(patt);

ind = 0;

while(i < l1)

{

if(text[i] == patt[j])

{

i++;

j++;

}

if(j == l2)

{

ans[ind++] = i - j;

j = lps[j-1];

}

else if(text[i] != patt[j])

{

if(j == 0)i++;

else j = lps[j-1];

}

}

}

**Z-algorithm:**

void zalgo()

{

LL i,l,r;

l = r = 0;

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

{

if(i > r)

{

l = r = i;

while(r < n && str[r] == str[r-l])r++;

z[i] = r - l;

r--;

}

else

{

LL k = i - l;

if(z[k] < r-i+1)

{

z[i] = z[k];

}

else

{

l = i;

while(r < n && str[r] == str[r-l])r++;

z[i] = r - l;

r--;

}

}

}

}

**Suffix Array:**

struct node

{

LL ind;

LL val[2];

bool operator < ( const node& p ) const

{

if(val[0] == p.val[0])

{

return val[1] < p.val[1];

}

else return val[0] < p.val[0];

}

};

node L[Max];

LL P[18][Max],D,S,len;

char str[Max];

LL LCP(LL x,LL y)

{

LL i,j,ans = 0;

if(x == y)return len - x;

for(i = D,j = S; j>=0 && x<len && y<len; i/=2,j--)

{

if(P[j][x] == P[j][y])

{

ans += i;

x += i;

y += i;

}

}

return ans;

}

node Lsrt[Max];

LL MX;

LL cnt[Max],taken[Max],spcl,spcltaken,sum[Max];

void Sort()

{

int i,j,idx,l;

Rep(i,len)

{

LL my = L[i].val[1];

if(my == -1)spcl++;

else cnt[my]++;

}

sum[0] = spcl;

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

{

sum[i] = sum[i-1] + cnt[i-1];

}

Rep(i,len)

{

LL my = L[i].val[1];

if(my == -1)

{

spcltaken++;

Lsrt[ spcltaken-1 ] = L[i];

}

else

{

taken[my]++;

Lsrt[ sum[my] + taken[my]-1 ] = L[i];

}

}

spcl = spcltaken = 0;

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

{

taken[i] = cnt[i] = 0;

}

Rep(i,len)

{

LL my = Lsrt[i].val[0];

if(my == -1)spcl++;

else cnt[my]++;

}

sum[0] = spcl;

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

{

sum[i] = sum[i-1] + cnt[i-1];

}

Rep(i,len)

{

LL my = Lsrt[i].val[0];

if(my == -1)

{

spcltaken++;

L[ spcltaken-1 ] = Lsrt[i];

}

else

{

taken[my]++;

L[ sum[my] + taken[my]-1 ] = Lsrt[i];

}

}

spcl = spcltaken = 0;

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

{

taken[i] = cnt[i] = 0;

}

}

void SA()

{

LL i,d,stp;

MX = 0;

Rep(i,len)

{

P[0][i] = str[i] - 'a';

MX = max(MX,P[0][i]);

}

for(d=1,stp = 1; d<len; d\*=2,stp++)

{

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

{

L[i].ind = i;

L[i].val[0] = P[stp - 1][i];

if(i+d < len)L[i].val[1] = P[stp - 1][i + d];

else L[i].val[1] = -1;

}

Sort(); ///use sort(L,L+len) instead for slower version

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

{

if(i > 0 && L[i].val[0] == L[i-1].val[0] && L[i].val[1] == L[i-1].val[1])

{

P[stp][ L[i].ind ] = P[stp][ L[i-1].ind ];

}

else P[stp][ L[i].ind ] = i;

MX = max(MX,P[stp][ L[i].ind ]);

}

}

D = d;

S = stp-1;

}

**Manachar:**

int d1[Max],n;

char s[Max],str[Max];

void manodd()

{

int l=0, r=-1;

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

{

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

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;

}

}

///(id+1)\*2 actual index, (id+1)\*2-1 is between (id-1) and id

void make()

{

int k = 0;

s[k++] = '$';

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

{

s[k++] = '#';

s[k++] = str[i];

}

s[k++] = '#';

s[k] = 0;

n = k;

}

**Palindromic Tree:**

const int MAXN = 105000;

struct Node

{

int next[26];

int len;

int sufflink;

int num;

int ed;

};

char s[MAXN];

Node tree[MAXN];

int num; // node 1 - root with len -1, node 2 - root with len 0

int suff; // max suffix palindrome

long long ans;

int len;

bool addLetter(int pos)

{

int cur = suff, curlen = 0;

int let = s[pos] - 'a';

while(true)

{

curlen = tree[cur].len;

if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])break;

cur = tree[cur].sufflink;

}

if (tree[cur].next[let])

{

suff = tree[cur].next[let];

return false;

}

num++;

suff = num;

tree[num].len = tree[cur].len + 2;

tree[cur].next[let] = num;

tree[num].ed = pos;

if (tree[num].len == 1)

{

tree[num].sufflink = 2;

tree[num].num = 1;

return true;

}

while (true)

{

cur = tree[cur].sufflink;

curlen = tree[cur].len;

if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])

{

tree[num].sufflink = tree[cur].next[let];

break;

}

}

tree[num].num = 1 + tree[tree[num].sufflink].num;

return true;

}

void initTree()

{

num = 2;

suff = 2;

tree[1].len = -1;

tree[1].sufflink = 1;

tree[2].len = 0;

tree[2].sufflink = 1;

}

int main()

{

len = strlen(s);

initTree();

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

{

addLetter(i);

ans += tree[suff].num;

}

return 0;

}

**Dynamic Programming:**

**LIS NlongN:**

LL a[Max],N,dp[Max],c[Max];

LL bin(LL len,LL x)

{

LL low = 1,high = len, mid, ans;

if( len == 0 || x > c[len] ) return len + 1;

while(low <= high)

{

mid = (low + high) / 2;

LL my = c[mid];

if( x <= my )

{

ans = mid;

high = mid - 1;

}

else low = mid + 1;

}

return ans;

}

LL LIS()

{

LL len = 0;

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

{

LL k = bin(len,a[i]);

c[k] = a[i];

dp[i] = k;

len = max(len,k);

}

return len;

}

**ConvexHull Trick:**

{normally we solve for 1 line then add another line according to the result(c)}

///line

struct line

{

LL m,c;

line() {};

line(LL \_m,LL \_c) { m = \_m; c = \_c;}

bool operator < (const line &p) const

{

if(m == p.m)return c > p.c;

else return m > p.m;

}

};

line ara[Max],st[Max];

LL n,s;

bool isbad(line l1,line l2,line l3)

{

return 1.0 \* (l3.c - l1.c) \* (l1.m - l2.m) < 1.0 \* (l2.c - l1.c) \* (l1.m - l3.m);

}

void add(line L)

{

while(s > 1)

{

if( isbad(st[s-2],st[s-1],L) )s--;

else break;

}

st[s++] = L;

}

LL f; ///is there decimal part

LL Intersection(LL idx1,LL idx2)

{

LL val = (st[idx2].c - st[idx1].c) / (st[idx1].m - st[idx2].m);

if( (st[idx2].c - st[idx1].c) % (st[idx1].m - st[idx2].m) )

{

LL a = (st[idx2].c - st[idx1].c);

if( a < 0 )a = -1;

else a = +1;

LL b = (st[idx1].m - st[idx2].m);

if( b < 0 )b = -1;

else b = +1;

if( (a\*b) < 0 )f = -1;

else f = +1;

}

else f = 0;

return val;

}

LL Find(LL x)

{

LL low = 0,high = s-1,mid,ans;

while(low <= high)

{

mid = (low + high) / 2;

LL my = INF;

if(mid != s-1)my = Intersection(mid,mid+1);

my \*= 2;

if(f)my++;

if(my >= 2\*x)

{

ans = mid;

high = mid - 1;

}

else low = mid + 1;

}

return st[ans].m \* x + st[ans].c;

}

**Divide and Conquer Trick:**

int Q(int x,int y)

{

///this is some random cost function

return cum[y][y] - cum[y][x-1] - cum[x-1][y] + cum[x-1][x-1];

}

int dp[805][Max];

void F(int d,int l,int r,int optl,int optr)

{

if(l > r)return;

int mid = (l + r) / 2;

int ret = INF,idx;

for(int i=optl;i<=optr;i++)

{

int my = dp[d-1][i] + Q(i+1,mid);

if(my < ret)

{

ret = my;

idx = i;

}

}

dp[d][mid] = ret;

if(l == r)return;

F(d,l,mid-1,optl,idx);

F(d,mid+1,r,idx,optr);

}

int main()

{

rep(i,n)dp[1][i] = Q(1,i);

for(i=2;i<=k;i++)F(i,1,n,1,n);

cout << dp[k][n];

return 0;

}

**Math:**

**Extended Euclid:**

struct node

{

LL g,x,y;

node() {}

node(LL \_g,LL \_x,LL \_y) { g = \_g; x = \_x; y = \_y; }

};

node xgcd(LL a,LL b)

{

if(b == 0) return node(a,1,0);

node tmp = xgcd(b,a % b);

return node(tmp.g, tmp.y, (tmp.g - a\*tmp.y)/b);

}

struct info

{

LL x,y;

info() {}

info(LL \_x, LL \_y) { x = \_x; y = \_y; }

};

bool generate\_a\_solution(LL a,LL b,LL c,info &ret)

{

if( a == 0 || b == 0 )

{

if( a + b == 0 ) { ret = info(0,0); return c == 0; }

if( a == 0 ) { ret = info(0,c/b); return c % b == 0; }

if( b == 0 ) { ret = info(c/a,0); return c % a == 0; }

}

///making cool

LL s1 = a / llabs(a); a = llabs(a);

LL s2 = b / llabs(b); b = llabs(b);

LL g = \_\_gcd(a,b);

if( c % g )return false; ///leave it while copying

LL val = c / g;

///done cooling

node my = xgcd(a,b);

LL x = my.x \* val \* s1;

LL y = my.y \* val \* s2;

ret = info(x,y);

assert( (s1\*a\*x + s2\*b\*y) == c );

return true;

}

LL keep\_x\_inrange(LL a,LL b,LL c,LL l,LL r,info &ret1,info &ret2)

{

info my;

bool f = generate\_a\_solution(a,b,c,my);

if(f == 0)return 0;

LL x = my.x; LL y = my.y;

///making cool

LL s1 = a / llabs(a); a = llabs(a);

LL s2 = b / llabs(b); b = llabs(b);

LL g = \_\_gcd(a,b);

LL val = c / g;

///done cooling

LL dx = b / g;

LL dy = a / g;

if( x < l )

{

LL need = l - x;

LL item = need / dx;

if( need % dx )item++;

x += (item \* dx);

y += (item \* dy) \* (s1 \* s2 \* -1);

}

else

{

LL need = x - l;

LL item = need / dx;

x -= (item \* dx);

y -= (item \* dy) \* (s1 \* s2 \* -1);

}

ret1 = info(x,y);

if( x > r )

{

LL need = x - r;

LL item = need / dx;

if( need % dx )item++;

x -= (item \* dx);

y -= (item \* dy) \* (s1 \* s2 \* -1);

}

else

{

LL need = r - x;

LL item = need / dx;

x += (item \* dx);

y += (item \* dy) \* (s1 \* s2 \* -1);

}

ret2 = info(x,y);

if( ret2.x < ret1.x )return 0;

return (ret2.x - ret1.x) / dx + 1;

}

LL keep\_y\_inrange(LL a,LL b,LL c,LL l,LL r,info &ret1,info &ret2)

{

info my;

bool f = generate\_a\_solution(a,b,c,my);

if(f == 0)return 0;

LL x = my.x; LL y = my.y;

///making cool

LL s1 = a / llabs(a); a = llabs(a);

LL s2 = b / llabs(b); b = llabs(b);

LL g = \_\_gcd(a,b);

LL val = c / g;

///done cooling

LL dx = b / g;

LL dy = a / g;

if( y < l )

{

LL need = l - y;

LL item = need / dy;

if( need % dy )item++;

y += (item \* dy);

x += (item \* dx) \* (s1 \* s2 \* -1);

}

else

{

LL need = y - l;

LL item = need / dy;

y -= (item \* dy);

x -= (item \* dx) \* (s1 \* s2 \* -1);

}

ret1 = info(x,y);

if( y > r )

{

LL need = y - r;

LL item = need / dy;

if( need % dy )item++;

y -= (item \* dy);

x -= (item \* dx) \* (s1 \* s2 \* -1);

}

else

{

LL need = r - y;

LL item = need / dy;

y += (item \* dy);

x += (item \* dx) \* (s1 \* s2 \* -1);

}

ret2 = info(x,y);

if(ret2.y < ret1.y)return 0;

return (ret2.y - ret1.y) / dy + 1;

}

LL keep\_xy\_inrange(LL a,LL b,LL c,LL x1,LL x2,LL y1,LL y2,info &ret1,info &ret2)

{

bool f = generate\_a\_solution(a,b,c,ret1);

if(f == 0)return 0;

if( a == 0 || b == 0 )

{

if( a + b == 0 ) return c == 0 ? (x2 - x1 + 1) \* (y2 - y1 + 1) : 0;

if( a == 0 ) return ( y1 <= ret1.y && ret1.y <= y2 ) ? (x2 - x1 + 1) : 0;

if( b == 0 ) return ( x1 <= ret1.x && ret1.x <= x2 ) ? (y2 - y1 + 1) : 0;

}

info retx1,retx2;

LL cx = keep\_x\_inrange(a,b,c,x1,x2,retx1,retx2);

if( cx == 0 )return 0;

LL st = retx1.y;

LL ed = retx2.y;

if(st > ed)swap(st,ed);

info rety1,rety2;

LL cy = keep\_y\_inrange(a,b,c,max(y1,st),min(y2,ed),rety1,rety2);

ret1 = rety1;

ret2 = rety2;

return cy;

}

**Gauss:**

#include <bits/stdc++.h>

#define Max 105

#define mem(ara,val) memset(ara,val,sizeof(ara))

#define eps 1e-10

using namespace std;

double a[Max][Max],b[Max][Max];

int n,m;

double Gauss()

{

int irow[Max],icol[Max],ipiv[Max];

mem(irow,0);

mem(icol,0);

mem(ipiv,0);

double det = 1.0;

int i,j,k;

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

{

int pj = -1, pk = -1;

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

{

if(!ipiv[j])

{

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

{

if(!ipiv[k])

{

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

{

pj = j;

pk = k;

}

}

}

}

}

if (fabs(a[pj][pk]) < eps)

{

cout << "Matrix is singular." << endl;

return -1.0;

}

ipiv[pk]++;

for(j=0; j<n; j++)swap(a[pj][j],a[pk][j]);

for(j=0; j<m; j++)swap(b[pj][j],b[pk][j]);

if (pj != pk) det \*= -1.0;

irow[i] = pj;

icol[i] = pk;

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

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

a[pk][pk] = 1.0;

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

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

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

{

if (p != pk)

{

c = a[p][pk];

a[p][pk] = 0;

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

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

}

}

}

for (int p = n-1; p >= 0; p--)

{

if (irow[p] != icol[p])

{

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

}

}

return det;

}

int main()

{

int i,j;

n = 4;

m = 2;

double A[4][4] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };

double B[4][2] = { {1,2},{4,3},{5,6},{8,7} };

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

{

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

}

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

{

for(j=0; j<m; j++)b[i][j] = B[i][j];

}

double ans = Gauss();

printf("det %f\n",ans);

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

{

for(j=0; j<n; j++)printf("%f ",a[i][j]);

puts("");

}

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

{

for(j=0; j<m; j++)printf("%f ",b[i][j]);

puts("");

}

return 0;

}

**CRT:**

LL Inverse(LL a,LL md)

{

return bigmod(a,md-2,md);

}

int main()

{

LL t,T,n,i,modulo[20],remainder[20];

sl(T);

rep(t,T)

{

sl(n);

LL M = 1;

rep(i,n)

{

sll(modulo[i],remainder[i]);

M \*= modulo[i];

}

LL tot = 0;

rep(i,n)

{

LL b = M / modulo[i];

LL bpr = Inverse(b,modulo[i]);

tot += (remainder[i] \* b \* bpr);

tot %= M;

}

printf("Case %lld: %lld\n",t,tot);

}

return 0;

}

**Data Structures:**

**Binary Indexed Tree:**

LL read(LL ind)

{

LL sum = 0;

while(ind > 0)

{

sum += tree[ind];

ind -= (ind & -ind);

}

return sum;

}

void update(LL ind,LL val)

{

while(ind <= mx)

{

tree[ind] += val;

ind += (ind & -ind);

}

}

LL readsingle(LL ind)

{

LL sum = tree[ind];

if(ind > 0)

{

LL z = ind - (ind & -ind);

ind--;

while(ind != z)

{

sum -= tree[ind];

ind -= (idn & -ind);

}

}

return sum;

}

**Disjoint Set Union:**

LL Find(LL r)

{

if(r == par[r])return r;

return par[r] = Find(par[r]);

}

void Union(LL x,LL y)

{

LL u = Find(x);

LL v = FInd(y);

if(u == v)return;///cycle detected

par[u] = v;

}

///initially everybody is parent of his own

**Heavy Light Decomposition:**

vector <int> g[Max];

int n;

int par[Max];///

bool visit[Max];///

int child[Max],special[Max];

void dfs(int u)

{

visit[u] = 1;

int len = sz(g[u]),i,mx = 0,idx = -1;

child[u] = 0;

Rep(i,len)

{

int v = g[u][i];

if(visit[v] == 0)

{

par[v] = u;

dfs(v);

if(child[v] > mx)

{

mx = child[v];

idx = v;

}

child[u] += child[v];

}

}

child[u]++;

special[u] = idx;

}

int head[Max]/\*\*/,tail[Max],dis[Max],path[Max],chaindx[Max],cn/\*\*/,sand/\*\*/;

void HLD(int u)

{

sand++;

if(head[cn] == -1)head[cn] = sand;

tail[cn] = sand;

dis[u] = sand;

path[sand] = u;

chaindx[sand] = cn;

if(special[u] != -1)HLD(special[u]);

int len = sz(g[u]),i;

Rep(i,len)

{

int v = g[u][i];

if(v == par[u] || v == special[u])continue;

cn++;

HLD(v);

}

}

**LCA:**

void dfs(LL u,LL dept)

{

L[u] = dept;

LL len = sz(g[u]),i;

Rep(i,len)

{

LL v = g[u][i];

if(par[u][0] != v)

{

par[v][0] = u;

dfs(v,dept+1);

}

}

}

void init()

{

col = (LL)(log10((double)(n-1))/log10(2.0));

mem(par,-1);

dfs(0,0);

LL i,j;

for(j=1;j<=col;j++)

{

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

{

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

{

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

}

}

}

}

LL query(LL a,LL b)

{

LL i;

if(L[a] < L[b])swap(a,b);

for(i=col;i>=0;i--)

{

if(L[a] - (1<<i) >= L[b])

{

a = par[a][i];

}

}

if(a == b)return a;

for(i=col;i>=0;i--)

{

if(par[a][i] != -1 && par[a][i] != par[b][i])

{

a = par[a][i];

b = par[b][i];

}

}

return par[a][0];

}

**SQRT Decomposition:**

///bucket size N

/// i'th bucket range ( (i-1) \* N + 1 ) to ( i \* N )

/// x is in the ( (x-1) / N + 1 )th bucket

int N,S,n,m,u;

int ara[Max],bra[Max];

void prework()

{

int i;

rep(i,n)bra[i] = ara[i];

N = sqrt(n);

S = n / N;

if(n % N)S++;

rep(i,S)

{

int l = (i-1) \* N + 1;

int r = min(n,i\*N);

sort(bra+l,bra+r+1);

}

}

///single update

inline void Update(int idx,int x)

{

int d = (idx - 1) / N + 1;

int l = (d-1) \* N + 1;

int r = min(n,d\*N);

ara[idx] = x;

for(int i=l;i<=r;i++)bra[i] = ara[i];

sort(bra+l,bra+r+1);

}

///how many numbers less than v in range [x,y]

inline int Q(int x,int y,int v)

{

int i;

int d1 = (x-1) / N + 1;

int d2 = (y-1) / N + 1;

int ret = 0;

for(i=d1+1;i<=d2-1;i++)

{

int l = (i-1) \* N + 1;

int r = i \* N;

int low = l,high = r,mid,ans = -1;

while(low <= high)

{

mid = (low + high) / 2;

int my = bra[mid];

if(my < v)

{

ans = mid;

low = mid + 1;

}

else high = mid - 1;

}

if(ans == -1)ret += 0;

else ret += (ans - l + 1);

}

if( d1 == d2 )

{

for(i=x;i<=y;i++)

{

if( ara[i] < v )ret++;

}

}

else

{

int r = min(n,d1 \* N);

for(i=x;i<=r;i++)

{

if( ara[i] < v )ret++;

}

int l = (d2 - 1) \* N + 1;

for(i=l;i<=y;i++)

{

if( ara[i] < v )ret++;

}

}

return ret;

}

**Mo's Algorithm:**

///bucket size N

/// i'th bucket range ( (i-1) \* N + 1 ) to ( i \* N )

/// x is in the ( (x-1) / N + 1 )th bucket

struct info

{

int x,y,idx;

bool operator < (const info &p) const

{

return y < p.y;

}

};

info q[Max];

vector <info> v[Max];

int N,n,m,ara[Max],S;

void prework()

{

N = sqrt(n);

S = n / N;

if(n % N)S++;

int i;

rep(i,m)

{

int b = ( q[i].x - 1 ) / N + 1;

v[b].pb( q[i] );

}

rep(i,S)sort(v[i].begin(),v[i].end());

}

int ans[Max],cnt[Max];

///most frequent value in the range

void F()

{

int i,j,k;

rep(i,S)

{

int l = (i-1) \* N + 1;

int r = min(n,i \* N);

int cur = r;

int mx = 0;

mem(cnt,0);

Rep(j,sz(v[i]))

{

info tmp = v[i][j];

for(k=cur+1;k<=tmp.y;k++)

{

cnt[ ara[k] ]++;

mx = max(mx,cnt[ ara[k] ]);

}

cur = max(cur,tmp.y);

int tmpmx = 0;

for(k=tmp.x;k<=min(r,tmp.y);k++)

{

cnt[ ara[k] ]++;

tmpmx = max(tmpmx,cnt[ ara[k] ]);

}

for(k=tmp.x;k<=min(r,tmp.y);k++)

{

cnt[ ara[k] ]--;

}

ans[ tmp.idx ] = max(tmpmx,mx);

}

}

}

**Treap:**

struct node

{

int val,prior,size;

struct node \*l,\*r;

void Show()

{

printf("address : %d\n",&val);

printf("val %d prior %d size %d\n",val,prior,size);

printf("left %d right %d\n",l,r);

puts("----------------------------");

}

};

typedef node\* pnode;

int SZ(pnode t)

{

return t ? t->size : 0;

}

void upd\_sz(pnode t)

{

if(t)t->size = SZ(t->l) + SZ(t->r) + 1;

}

void Split(pnode t,pnode &l,pnode &r,int key)

{

if(!t)l = r = 0;

else if(t->val <= key)Split(t->r,t->r,r,key),l = t; //elem=key comes in l

else Split(t->l,l,t->l,key),r = t;

upd\_sz(t);

}

void Merge(pnode &t,pnode l,pnode r)

{

if(!l || !r)t = l ? l : r;

else if(l->prior > r->prior)Merge(l->r,l->r,r),t = l;

else Merge(r->l,l,r->l),t = r;

upd\_sz(t);

}

void Insert(pnode &t,pnode it)

{

if(!t)t = it;

else if(it->prior > t->prior)Split(t,it->l,it->r,it->val),t = it;

else Insert(t->val <= it->val ? t->r : t->l,it);

upd\_sz(t);

}

void Erase(pnode &t,int key)

{

if(!t)return;

else if(t->val == key)

{

pnode temp = t;

Merge(t,t->l,t->r);

free(temp);

}

else Erase(t->val < key ? t->r : t->l,key);

upd\_sz(t);

}

pnode init(int val)

{

pnode ret = new node();

ret->val = val;

ret->size = 1;

ret->prior = rand();

ret->l = ret->r = 0;

return ret;

}

void dfs(pnode t)

{

if(!t)return;

t->Show();

if(t->l)dfs(t->l);

if(t->r)dfs(t->r);

}

**Implicit Treap:**

struct node

{

int prior,size;

int val; ///value stored in the array

int sum; ///whatever info you want to maintain in segtree for each node

int lazy; ///whatever lazy update you want to do

struct node \*l,\*r;

};

typedef node\* pnode;

int SZ(pnode t)

{

return t ? t->size : 0;

}

void upd\_sz(pnode t)

{

if(t)t->size = SZ(t->l) + SZ(t->r) + 1;

}

void lazy(pnode t)

{

if(!t || !t->lazy)return;

t->val += t->lazy; ///operation of lazy

t->sum += t->lazy \* SZ(t);

if(t->l)t->l->lazy += t->lazy;///propagate lazy

if(t->r)t->r->lazy += t->lazy;

t->lazy = 0;

}

void reset(pnode t)

{

if(t)t->sum = t->val;///no need to reset lazy coz when we call this lazy would itself be propagated

}

void combine(pnode& t,pnode l,pnode r)

{

///combining two ranges of segtree

if(!l || !r)return void(t = l ? l : r);

t->sum = l->sum + r->sum;

}

void operation(pnode t)

{

///operation of segtree

if(!t)return;

reset(t);///reset the value of current node assuming it now represents a single element of the array

lazy(t->l);lazy(t->r);///imp:propagate lazy before combining t->l,t->r;

combine(t,t->l,t);

combine(t,t,t->r);

}

void Split(pnode t,pnode &l,pnode &r,int pos,int add=0)

{

if(!t)return void(l = r = NULL);

lazy(t);

int curr\_pos = add + SZ(t->l) + 1;

if(curr\_pos <= pos)Split(t->r,t->r,r,pos,curr\_pos),l = t;///element at pos goes to left subtree(l)

else Split(t->l,l,t->l,pos,add),r = t;

upd\_sz(t);

operation(t);

}

void Merge(pnode &t,pnode l,pnode r)

{

///l->leftarray,r->rightarray,t->resulting array

lazy(l);lazy(r);

if(!l || !r) t = l ? l : r;

else if(l->prior > r->prior)Merge(l->r,l->r,r),t = l;

else Merge(r->l,l,r->l),t = r;

upd\_sz(t);

operation(t);

}

pnode init(int val)

{

pnode ret = new node();

ret->prior = rand();

ret->size = 1;

ret->val = val;

ret->sum = val;

ret->lazy = 0;

return ret;

}

void ArrayInsert(pnode &t,pnode it,int pos)

{

pnode L,R,tmp;

Split(t,L,R,pos);

Merge(L,L,it);

Merge(t,L,R);

}

void ArrayDelete(pnode &t,int pos)

{

pnode L,R,tmp;

Split(t,L,R,pos);

Split(L,L,tmp,pos-1);

Merge(t,L,R);

}

int ArrayValue(pnode t,int pos)

{

int cur = 1 + SZ(t->l);

if(cur == pos)return t->val;

if(cur < pos)return ArrayValue(t->r,pos-cur);

else return ArrayValue(t->l,pos);

}

int query(pnode t,int l,int r)

{

///[l,r]

pnode L,mid,R;

Split(t,L,mid,l-1);

Split(mid,t,R,r-l+1);///note: r-l!!

int ans = t->sum;

Merge(mid,L,t);

Merge(t,mid,R);

return ans;

}

void update(pnode t,int l,int r,int val)

{

///[l,r]

pnode L,mid,R;

Split(t,L,mid,l-1);

Split(mid,t,R,r-l+1);///note: r-l!!

t->lazy += val; ///lazy\_update

Merge(mid,L,t);

Merge(t,mid,R);

}

**Misc and Geometry:**

**Postfix and Result:**

struct info

{

bool opt;

char c;

LL val;

void make(bool x,char y,LL z)

{

opt = x;c = y;val = z;

}

};

info ara[Max];

LL L;

stack <info> st;

vector <info> q;

LL precedence(info A,info B)

{

if(A.c == '+' || A.c == '-')

{

if(B.c == '+' || B.c == '-')return 0;

return -1;

}

else

{

if(B.c == '\*' || B.c == '/')return 0;

return +1;

}

}

void postfix()

{

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

q.clear();

info tmp;

tmp.make(1,')',0);

ara[L++] = tmp;

tmp.make(1,'(',0);

st.push(tmp);

for(LL i=0;i<L;i++)

{

if(ara[i].opt == 0)q.pb(ara[i]);

else

{

if(ara[i].c == '(')st.push(ara[i]);

else if(ara[i].c == ')')

{

while(1)

{

tmp = st.top();

st.pop();

if(tmp.c == '(')break;

q.pb(tmp);

}

}

else

{

while(!st.empty())

{

tmp = st.top();

if(tmp.c == '(' || tmp.c == ')')break;

LL my = precedence(ara[i],tmp);

if(my <= 0)

{

q.pb(tmp);

st.pop();

}

else break;

}

st.push(ara[i]);

}

}

}

}

LL result()

{

info tmp;

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

LL len = sz(q),i;

Rep(i,len)

{

info my = q[i];

if(my.opt == 0)st.push(my);

else

{

info A = st.top();st.pop();

info B = st.top();st.pop();

LL val;

if(my.c == '+')val = B.val + A.val;

if(my.c == '-')val = B.val - A.val;

if(my.c == '\*')val = B.val \* A.val;

if(my.c == '/')val = B.val / A.val;

tmp.make(0,0,val);

st.push(tmp);

}

}

return st.top().val;

}

**String X String**

#include <stdio.h>

#include <string.h>

int main()

{

int c,len,i,f,k,h,s,x,j;

char a[2000],b[2000],ch[2000],num[4000];

while(scanf("%s %s",a,b)!=EOF)

{

c=0,j=0;

memset(ch,0,sizeof(ch));

len=strlen(a);

if(strlen(a)<strlen(b))

{

len=strlen(b);

for(i=0; i<len-strlen(a); i++)ch[i]='0';

strcat(ch,a);

strcpy(a,ch);

}

else

{

for(i=0; i<len-strlen(b); i++)ch[i]='0';

strcat(ch,b);

strcpy(b,ch);

}

memset(ch,0,sizeof(ch));

for(h=0,x=len-1,k=2\*len-2,i=1; i<=len; i++,k--)

{

for(s=0,f=0; f<i; f++)

{

s+=(a[x-f]-48)\*(b[k-x+f]-48);

}

s+=h;

ch[j++]=s%10+48;

h=s/10;

}

for(i=x; i>0; i--,k--)

{

for(s=0,f=0; f<i; f++)

{

s+=(a[f]-48)\*(b[k-f]-48);

}

s+=h;

ch[j++]=s%10+48;

h=s/10;

}

for(i=0; h>0; i++)

{

ch[j++]=h%10+48;

h=h/10;

}

memset(num,0,sizeof(num));

for(h=0,c=0,i=0; i<j; i++)

{

if(ch[j-1-i]=='0' && c==0)continue;

num[h++]=ch[j-i-1];

c++;

}

if(strlen(num)==0)printf("0\n");

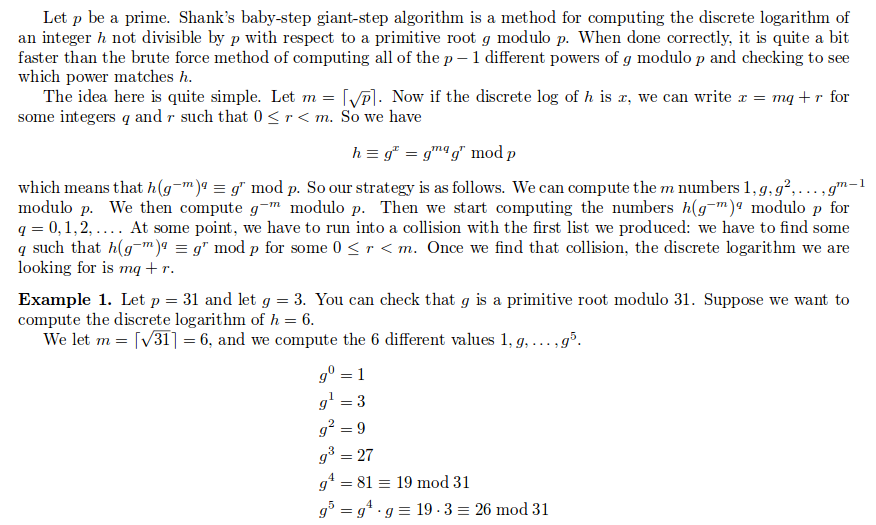
else printf("%s\n",num);

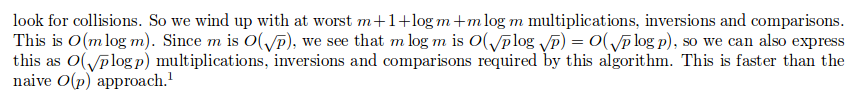
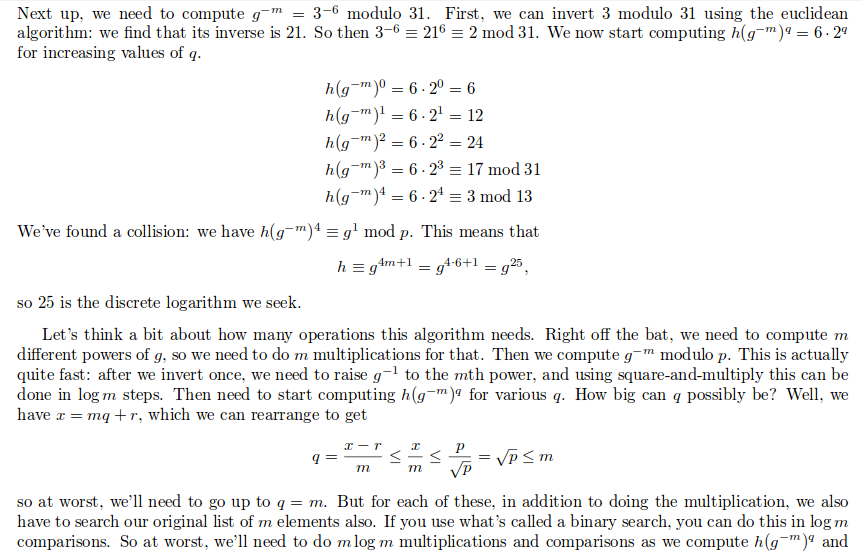
}

return 0;

}

**Shank's baby-step giant-step algorithm:**





**Stable Marriage Problem:**

function stableMatching

{

Initialize all m ∈ M and w ∈ W to free

while ∃ free man m who still has a woman w to propose to

{

w = m's highest ranked such woman to whom he has not yet proposed

if w is free

(m, w) become engaged

else some pair (m', w) already exists

if w prefers m to m'

(m, w) become engaged

m' becomes free

else (m', w) remain engaged

}

}

**Convex Hull(Minimized):**

struct Point

{

int x,y;

};

Point p0,points[Max],ara[Max];

int n,id;

LL distSq(Point p1, Point p2){return (LL)(p1.x - p2.x)\*(LL)(p1.x - p2.x) + (LL)(p1.y - p2.y)\*(LL)(p1.y - p2.y);}

///1 CCW,-1 CW,0 CL

int orientation(Point A,Point B,Point C)

{

LL val = (LL)A.x\*(LL)(B.y-C.y) - (LL)A.y\*(LL)(B.x-C.x) + (LL)B.x\*(LL)C.y - (LL)C.x\*(LL)B.y;

if(val > 0)val = 1;

else if(val < 0)val = -1;

return val;

}

bool cmp(Point A,Point B)

{

int o = orientation(p0,A,B);

if(o == 0)return distSq(p0,B) > distSq(p0,A);

else return o == 1;

}

stack <Point> S;

Point nextToTop()

{

Point my = S.top();

S.pop();

Point ret = S.top();

S.push(my);

return ret;

}

bool convexHull()

{

int i,j;

int mn = points[0].y,idx = 0;

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

{

int y = points[i].y;

if ((y < mn) || (mn == y && points[i].x < points[idx].x))

{

mn = points[i].y;

idx = i;

}

}

swap(points[0], points[idx]);

p0 = points[0];

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

int m = 1;

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

{

while(i<n-1 && orientation(p0,points[i],points[i+1]) == 0)i++;

points[m] = points[i];

m++;

}

if(m < 3)return false;

S.push(points[0]);

S.push(points[1]);

S.push(points[2]);

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

{

while (orientation(nextToTop(),S.top(),points[i]) != 1)S.pop();

S.push(points[i]);

}

id = 0;

while (!S.empty())

{

Point my = S.top();

S.pop();

ara[id++] = my;

}

return true;

}

**Convex Hull(Maximized):**

struct Point

{

LL x,y,idx;

};

Point p0,points[Max],ara[Max],tp[Max];

LL n,id;

LL distSq(Point p1, Point p2)

{

return (LL)(p1.x - p2.x)\*(LL)(p1.x - p2.x) + (LL)(p1.y - p2.y)\*(LL)(p1.y - p2.y);

}

///1 CCW,-1 CW,0 CL

LL orientation(Point A,Point B,Point C)

{

LL val = (LL)A.x\*(LL)(B.y-C.y) - (LL)A.y\*(LL)(B.x-C.x) + (LL)B.x\*(LL)C.y - (LL)C.x\*(LL)B.y;

if(val > 0)val = 1;

else if(val < 0)val = -1;

return val;

}

bool cmp(Point A,Point B)

{

LL o = orientation(p0,A,B);

if(o == 0)return distSq(p0,B) > distSq(p0,A);

else return o == 1;

}

stack <Point> S;

Point nextToTop()

{

Point my = S.top();

S.pop();

Point ret = S.top();

S.push(my);

return ret;

}

bool convexHull()

{

LL i,j;

LL mn = points[0].y,idx = 0;

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

{

LL y = points[i].y;

if ((y < mn) || (mn == y && points[i].x < points[idx].x))

{

mn = points[i].y;

idx = i;

}

}

swap(points[0], points[idx]);

p0 = points[0];

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

LL cur = 0;

for(i=n-1;i>=1;i--)

{

cur = i;

if( orientation(p0,points[i-1],points[i]) )break;

}

LL k = 0;

for(i=n-1;i>=cur;i--)tp[k++] = points[i];

for(i=cur;i<n;i++)points[i] = tp[i-cur];

LL m = 1;

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

{

while(i<n-1 && orientation(p0,points[i],points[i+1]) == 0)i++;

m++;

}

if(m < 3)return false;

m = n;

S.push(points[0]);

S.push(points[1]);

S.push(points[2]);

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

{

while (orientation(nextToTop(),S.top(),points[i]) == -1)S.pop();

S.push(points[i]);

}

id = 0;

while (!S.empty())

{

Point my = S.top();

S.pop();

ara[id++] = my;

}

return true;

}

**Convex Hull Diameter:**

LL RotatingCalliper()

{

LL id,j = 0,mx = 0,i;

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

{

LL curdis = distSq(ara[i],ara[j]);

while(1)

{

LL nxtdis = distSq(ara[i],ara[ (j+1) % id ]);

if(nxtdis >= curdis)

{

j++;

if(j == id)j = 0;

curdis = nxtdis;

}

else break;

}

mx = max(mx,curdis);

}

return mx;

}

LL ter(LL idx)

{

LL low = idx+1,high = id-1,mid,ans = -1;

while(low <= high)

{

LL a = (2 \* low + high) / 3;

LL b = (low + 2 \* high + 1) / 3;

LL da = distSq(ara[idx],ara[a]);

LL db = distSq(ara[idx],ara[b]);

if(da < db)

{

ans = b;

low = a + 1;

}

else

{

ans = a;

high = b - 1;

}

}

return ans;

}

LL F()

{

LL i,mx = 0;

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

{

LL idx = ter(i);

if(idx == -1)continue;

LL d = distSq(ara[i],ara[idx]);

mx = max(mx,d);

}

return mx;

}

**Closest Pair:**

struct point

{

double x,y;

};

point p[Max],s[Max];

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

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

double Distance(point A,point B){ return sqrt( (A.x-B.x)\*(A.x-B.x) + (A.y-B.y)\*(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 ,ns = 0;

k = (k1 + k2) / 2;

d = closestPair(k1 , k);

d2 = closestPair(k+1 , k2);

if(d > d2) d = d2;

for(i = k1; i<=k2 ; i++)

{

if( fabs( p[i].x - p[k].x) <= d )s[ ns++ ] = p[i];

}

sort(s, s+ns, sortY);

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

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

{

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

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

}

}

return d;

}

**Utilities:**

**structures:**

struct point

{

double x,y;

point() {}

point(double \_x,double \_y){ x = \_x; y = \_y; }

};

struct line

{

double a,b,c;

line() {}

line(point p1,point p2)

{

a = p1.y - p2.y;

b = p2.x - p1.x;

c = p1.x \* p2.y - p2.x \* p1.y;

}

line(double \_a,double \_b,double \_c)

{

a = \_a;

b = \_b;

c = \_c;

}

double online(point p)

{

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

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

}

void extend(point p,double s,point &p1,point &p2)

{

if( b == 0 )

{

p1 = point(p.x,p.y+s);

p2 = point(p.x,p.y-s);

}

else

{

double dx = fabs(s\*b) / sqrt( a\*a + b\*b );

double dy = dx \* (-a/b);

p1 = point(p.x+dx,p.y+dy);

p2 = point(p.x-dx,p.y-dy);

}

}

};

struct segment

{

point A,B;

segment() {}

segment(point \_A,point \_B) { A = \_A; B = \_B; }

};

struct circle

{

point center;

double r;

circle() {}

circle(point \_c,double \_r){ center = \_c; r = \_r; }

};

struct Vector

{

point A,B;

Vector() {}

Vector(point \_A,point \_B) { A = \_A; B = \_B; }

Vector norm()

{

return Vector( point(0,0), point(B.x-A.x,B.y-A.y) );

}

Vector add(Vector V)

{

return Vector( A , point( B.x+V.norm().B.x, B.y+V.norm().B.y) );

}

Vector subtract(Vector V)

{

return Vector( A , point( B.x-V.norm().B.x, B.y-V.norm().B.y) );

}

};

**Equality:**

bool eq(double a,double b){ return fabs(a-b) < eps; }

bool lesseq(double x,double y) { if(x < y)return true; return eq(x,y); }

bool greateq(double x,double y) { if(x > y)return true; return eq(x,y); }

**Point Point Distance:**

double Distance(point A,point B){ return sqrt( (A.x-B.x)\*(A.x-B.x) + (A.y-B.y)\*(A.y-B.y) ); }

double distSq(point p1, point p2){return (p1.x - p2.x)\*(p1.x - p2.x) + (p1.y - p2.y)\*(p1.y - p2.y);}

**Orientation:**

///1 ccw -1 cw 0 cc

int orientation(point A,point B,point C)

{

double val = A.x\*(B.y-C.y) - A.y\*(B.x-C.x) + B.x\*C.y - C.x\*B.y;

int ret = 0;

if(val > 0.0)ret = 1;

else if(val < 0.0)ret = -1;

return ret;

}

**Q onsegment PR:**

///if q is on segment pr

bool onSegment(point p, point q, point r)

{

if( orientation(p,r,q) )return false;

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

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

return false;

}

bool onSegment(point p,point q,point r)

{

if( orientation(p,r,q) )return false;

if( lesseq( q.x,max(p.x,r.x) ) && greateq( q.x,min(p.x,r.x) ) &&

lesseq( q.y,max(p.y,r.y) ) && greateq( q.y,min(p.y,r.y) ) )return true;

return false;

}

**Line Line Intersection:**

bool intersection(line L1,line L2,point &p)

{

double d = L1.a \* L2.b - L1.b \* L2.a;

if( eq (d,0.0) )return false;

p.x = ( L1.b \* L2.c - L2.b \* L1.c ) / d;

p.y = ( L1.c \* L2.a - L2.c \* L1.a ) / d;

return true;

}

**Triangle Area:**

double TriangleArea(double a,double b,double c)

{

double s = (a + b + c) / 2.0;

return sqrt( s\*(s-a)\*(s-b)\*(s-c) );

}

double TriangleArea(point a,point b,point c)

{

return TriangleArea(dist(a,b),dist(b,c),dist(c,a));

}

bool point\_inside\_triange(point a,point b,point c,point p)

{

return eq( TriangleArea(a,b,p)+TriangleArea(b,c,p)+TriangleArea(c,a,p) , TriangleArea(a,b,c) );

}

double cap(double r,double x)

{

double theta = asin( x / (2\*r) ); ///half theta

return r\*r\*theta - TriangeArea(r,r,x);

}

**Polygon Area:**

double PolygonArea(point p[],int s)

{

double ret = 0;

for(int i=0,j=s-1;i<s;j=i++)ret += p[j].x \* p[i].y - p[j].y \* p[i].x;

return fabs(ret)/2;

}

**Centroid of a polygon:**

point centroid(point p[],int s)

{

double ar = Signed\_PolygonArea(p,s);

double x = 0.0, y = 0.0;

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

{

int j = (i + 1) % s;

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

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

}

return point( x / (6\*ar) , y / (6\*ar) );

}

**Angle:**

double Angle(point A,point B,point C)

{

double a = distSq(B,C);

double b = distSq(C,A);

double c = distSq(A,B);

double lob = (double)(a + c - b);

double hor = 2.0 \* sqrt( (double)a ) \* sqrt( (double)c );

return acos(lob/hor);

}

double Angle(double a,double b,double c)

{

double lob = (double)(a\*a + c\*c - b\*b);

double hor = 2.0 \* a \* b;

return acos(lob/hor);

}

**Perpendicular Line:**

line perpendicularline(line L,point p)

{

line ret;

ret.a = L.b;

ret.b = -L.a;

ret.c = -ret.a \* p.x - ret.b \* p.y;

return ret;

}

**Point Line Distance:**

double Distance(point p,line L)

{

return fabs( (L.a\*p.x + L.b\*p.y + L.c) / sqrt( L.a\*L.a + L.b\*L.b ) );

}

**Point Segment Distance:**

double Distance(point p,segment s)

{

line L1 = line(s.A,s.B);

line L2 = perpendicularline(L1,p);

point p1;

if( intersection(L1,L2,p1) )

{

if( eq( Distance(s.A,p1)+Distance(s.B,p1), Distance(s.A,s.B) ) )return Distance(p,p1);

}

return min( Distance(p,s.A),Distance(p,s.B) );

}

**Line Line Distance:**

double Distance(line L1,line L2)

{

double d = fabs(L1.a \* L2.b - L1.b \* L2.a);

if( fabs(L1.a \* L2.b - L1.b \* L2.a) < eps )

{

if( fabs(L2.b) > eps )return Distance( point(0.0,-L2.c/L2.b) , L1 );

else return Distance( point(-L2.c/L2.a,0.0) , L1 );

}

else return 0.0;

}

**Line Segment Distance:**

double Distance(line L,segment S)

{

double val1 = L.online(S.A);

double val2 = L.online(S.B);

if( fabs(val1) < eps || fabs(val2) < eps )return 0.0;

if(val1 \* val2 < eps)return 0.0;

return min( Distance(S.A,L) , Distance(S.B,L) );

}

**Circle Circle Intersection Area:**

double intersectionArea2C( circle C1, circle C2 ) {

C2.center.x = Distance( C1.center, C2.center );

C1.center.x = C1.center.y = C2.center.y = 0;

if( C1.r < C2.center.x - C2.r + eps ) return 0.0;

if( -C1.r + eps > C2.center.x - C2.r ) return PI \* C1.r \* C1.r;

if( C1.r + eps > C2.center.x + C2.r ) return PI \* C2.r \* C2.r;

double c, CAD, CBD, ret;

c = C2.center.x;

CAD = 2 \* acos( (C1.r \* C1.r + c \* c - C2.r \* C2.r) / (2 \* C1.r \* c) );

CBD = 2 \* acos( (C2.r \* C2.r + c \* c - C1.r \* C1.r) / (2 \* C2.r \* c) );

ret = C1.r \* C1.r \* ( CAD - sin( CAD ) ) + C2.r \* C2.r \* ( CBD - sin ( CBD ) );

return 0.5 \* ret;

}

**Circum Circle:**

double CircumCircleRadius(point A,point B,point C)

{

double a = Distance(B,C);

double b = Distance(C,A);

double c = Distance(A,B);

return (a\*b\*c) / sqrt( (a+b+c)\*(b+c-a)\*(c+a-b)\*(a+b-c) );

}

point CircumCircleCenter(point A,point B,point C)

{

double a = B.x - A.x;

double b = B.y - A.y;

double c = C.x - A.x;

double d = C.y - A.y;

double e = a \* (A.x + B.x) + b \* (A.y + B.y);

double f = c \* (A.x + C.x) + d \* (A.y + C.y);

double g = 2.0 \* (a \* (C.y - B.y) - b \* (C.x - B.x) );

point ret;

ret.x = (d \* e - b \* f) / g;

ret.y = (a \* f - c \* e) / g;

return ret;

}

///OGH: 2OG = GH

double distance\_between\_centroid\_and\_orthocenter(double a,double b,double c)

{

double r = CircumCircleRadius(a,b,c);

return sqrt( 4 \* r \* r - ( 4 \* (a\*a + b\*b + c\*c) ) / 9 );

}

double distance\_between\_centroid\_and\_circumcenter(double a,double b,double c)

{

double r = CircumCircleRadius(a,b,c);

return sqrt( r \* r - (a\*a + b\*b + c\*c) / 9 );

}

double distance\_between\_orthocenter\_and\_circumcenter(double a,double b,double c)

{

double r = CircumCircleRadius(a,b,c);

return sqrt( 9 \* r \* r - (a\*a + b\*b + c\*c) );

}

**Circle Line Intersection:**

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;

}

**Circle Circle Intersection:**

bool intersection(circle C1,circle C2,point &p1,point &p2)

{

if( Distance(C1.center,C2.center) > C1.r + C2.r + eps )return false;

if( min(C1.r,C2.r) + Distance(C1.center,C2.center) + eps < max(C1.r,C2.r) )return false;

double a = C2.center.x - C1.center.x;

double b = C2.center.y - C1.center.y;

double c = a\*a + b\*b + C1.r\*C1.r - C2.r\*C2.r;

a \*= -2.0;

b \*= -2.0;

line L = line(a,b,c);

point center = point(0.0,0.0);

circle C = circle(center,C1.r );

bool my = intersection(C,L,p1,p2);

if(my)

{

p1.x += C1.center.x;

p2.x += C1.center.x;

p1.y += C1.center.y;

p2.y += C1.center.y;

return true;

}

else return false;

}

**Segment Segment Intersection:**

bool intersection(segment S1,segment S2)

{

point p1,q1,p2,q2;

p1 = S1.A; q1 = S1.B;

p2 = S2.A; q2 = S2.B;

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

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

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

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

if (o1 != o2 && o3 != o4)return true;

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

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

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

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

return false;

}

**Segment Segment Distance:**

double Distance(segment S1,segment S2)

{

if( intersection(S1,S2) )return 0.0;

double ret = Distance(S1.A,S2);

ret= min(ret,Distance(S1.B,S2));

ret= min(ret,Distance(S2.A,S1));

ret= min(ret,Distance(S2.B,S1));

return ret;

}

**Directed MST:**

#define MAXNODE 1005

#define MAXEDGE 10005

struct Edge {

int u, v;

int dis;

Edge() {}

Edge(int \_u, int \_v, int \_dis) { u = \_u; v = \_v; dis = \_dis; }

};

int n, m;

Edge edges[MAXEDGE];

int vis[MAXNODE];

int pre[MAXNODE];

int id[MAXNODE];

int in[MAXNODE];

void init(int \_n) { n = \_n; m = 0; }

void AddEdge(int u, int v, int dis) { edges[m++] = Edge(u, v, dis); }

///0 based indexing

int directedMST(int root)

{

int ans = 0;

while(1)

{

for (int i = 0; i < n; i++) in[i] = INF;

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

{

int u = edges[i].u;

int v = edges[i].v;

/// find the smallest edge, delete the ring

if(edges[i].dis < in[v] && u != v)

{

in[v] = edges[i].dis;

pre[v] = u;

}

}

/// If there is no minimum entry, it means that the point is not connected, the smallest tree does not fail

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

{

if (i == root) continue;

if (in[i] == INF) return -1;

}

int cnt = 0;/// record shrink points

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

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

in[root] = 0;/// tree root can not have edge

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

{

ans += in[i];

int v = i;

/// find the ring

while (vis[v] != i && id[v] == -1 && v != root)

{

vis[v] = i;

v = pre[v];

}

/// find the ring

if (v != root && id[v] == -1)

{

for (int u = pre[v]; u != v; u = pre[u]) id[u] = cnt;

id[v] = cnt++;

}

}

/// If there is no self-loop, said the smallest tree shape to succeed

if (cnt == 0) break;

/// find those that are not self-looped and re-mark those points

for (int i = 0; i < n; i++) if (id[i] == -1) id[i] = cnt++;

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

{

int v = edges[i].v;

edges[i].v = id[edges[i].v];

edges[i].u = id[edges[i].u];

if (edges[i].u != edges[i].v) edges[i].dis -= in[v];

}

/// After the end of the point, the number of points on the side

n = cnt;

root = id[root];

}

return ans;

}

**Simplex:**

#define MAXC 505

#define MAXV 505

#define EPS 1e-13

#define MINIMIZE -1

#define MAXIMIZE +1

#define LESSEQ -1

#define EQUAL 0

#define GREATEQ 1

#define INFEASIBLE -1

#define UNBOUNDED 666

/\*\*\*

1. Simplex Algorithm for Linear Programming

2. Maximize or minimize f0\*x0 + f1\*x1 + f2\*x2 + ... + fn-1\*xn-1 subject to some constraints

3. Constraints are of the form, c0x0 + c1x1 + c2x2 + ... + cn-1xn-1 (<= or >= or =) lim

4. m is the number of constraints indexed from 1 to m, and n is the number of variables indexed from 0 to n-1

5. ar[0] contains the objective function f, and ar[1] to ar[m] contains the constraints, ar[i][n] = lim\_i

6. It is assumed that all variables satisfies non-negativity constraint, i.e, xi >= 0

7. If non-negativity constraint is not desired for a variable x, replace each occurrence

by difference of two new variables r1 and r2 (where r1 >= 0 and r2 >= 0, handled automatically by simplex).

That is, replace every x by r1 - r2 (Number of variables increases by one, -x, +r1, +r2)

8. solution\_flag = INFEASIBLE if no solution is possible and UNBOUNDED if no finite solution is possible

9. Returns the maximum/minimum value of the linear equation satisfying all constraints otherwise

10. After successful completion, val[] contains the values of x0, x1 .... xn for the optimal value returned

\*\*\* If ABS(X) <= M in constraints, Replace with X <= M and -X <= M

\*\*\* Fractional LP:

max/min

3x1 + 2x2 + 4x3 + 6

-------------------

3x1 + 3x2 + 2x3 + 5

s.t. 2x1 + 3x2 + 5x3 ≥ 23

3x2 + 5x2 + 4x3 <= 30

x1, x2, x3 ≥ 0

Replace with:

max/min

3y1 + 2y2 + 4y3 + 6t

s.t. 3y1 + 3y2 + 2y3 + 5t = 1

2y1 + 3y2 + 53 - 23t ≥ 0

3y1 + 5y2 + 4y3- 30t <= 0

y1, y2, y3, t ≥ 0

\*\*\*/

namespace lp

{

long double val[MAXV], ar[MAXC][MAXV];

int m, n, solution\_flag, minmax\_flag, basis[MAXC], index[MAXV];

/// nvars = number of variables, f = objective function, flag = MINIMIZE or MAXIMIZE

inline void init(int nvars, long double f[], int flag){

solution\_flag = 0;

ar[0][nvars] = 0.0;

m = 0, n = nvars, minmax\_flag = flag;

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

ar[0][i] = f[i] \* minmax\_flag; /// Negating sign of objective function when minimizing

}

}

/// C[] = co-efficients of the constraints (LHS), lim = limit in RHS

/// cmp = EQUAL for C[] = lim, LESSEQ for C[] <= lim, GREATEQ for C[] >= lim

inline void add\_constraint(long double C[], long double lim, int cmp){

m++, cmp \*= -1;

if (cmp == 0){

for (int i = 0; i < n; i++) ar[m][i] = C[i];

ar[m++][n] = lim;

for (int i = 0; i < n; i++) ar[m][i] = -C[i];

ar[m][n] = -lim;

}

else{

for (int i = 0; i < n; i++) ar[m][i] = C[i] \* cmp;

ar[m][n] = lim \* cmp;

}

}

inline void init(){ /// Initialization

for (int i = 0; i <= m; i++) basis[i] = -i;

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

ar[0][j] = -ar[0][j], index[j] = j, val[j] = 0;

}

}

inline void pivot(int m, int n, int a, int b){ /// Pivoting and exchanging a non-basic variable with a basic variable

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

if (i != a){

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

if (j != b){

ar[i][j] -= (ar[i][b] \* ar[a][j]) / ar[a][b];

}

}

}

}

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

if (j != b) ar[a][j] /= ar[a][b];

}

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

if (i != a) ar[i][b] = -ar[i][b] / ar[a][b];

}

ar[a][b] = 1.0 / ar[a][b];

swap(basis[a], index[b]);

}

inline long double solve(){ /// simplex core

init();

int i, j, k, l;

for (; ;){

for (i = 1, k = 1; i <= m; i++){

if ((ar[i][n] < ar[k][n]) || (ar[i][n] == ar[k][n] && basis[i] < basis[k] && (rand() & 1))) k = i;

}

if (ar[k][n] >= -EPS) break;

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

if ((ar[k][j] < (ar[k][l] - EPS)) || (ar[k][j] < (ar[k][l] - EPS) && index[i] < index[j] && (rand() & 1))){

l = j;

}

}

if (ar[k][l] >= -EPS){

solution\_flag = INFEASIBLE; /// No solution is possible

return -1.0;

}

pivot(m, n, k, l);

}

for (; ;){

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

if ((ar[0][j] < ar[0][l]) || (ar[0][j] == ar[0][l] && index[j] < index[l] && (rand() & 1))) l = j;

}

if (ar[0][l] > -EPS) break;

for (i = 1, k = 0; i <= m; i++){

if (ar[i][l] > EPS && (!k || ar[i][n] / ar[i][l] < ar[k][n] / ar[k][l] - EPS || (ar[i][n] / ar[i][l] < ar[k][n] / ar[k][l] + EPS && basis[i] < basis[k]))){

k = i;

}

}

if (ar[k][l] <= EPS){

solution\_flag = UNBOUNDED; /// Solution is infinity, no finite solution exists

return -666.0;

}

pivot(m, n, k, l);

}

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

if (basis[i] >= 0) val[basis[i]] = ar[i][n];

}

solution\_flag = 1; /// Successful completion

return (ar[0][n] \* minmax\_flag); /// Negate the output for MINIMIZE since the objective function was negated

}

}

**Egyptian Factorization:**

vector <LL> v;

void Egyptian(LL lob,LL hor)

{

if(lob == 0 || hor == 0)return;

LL g = \_\_gcd(lob,hor);

lob /= g;

hor /= g;

if(lob == 1)

{

v.pb(hor);

return;

}

LL my = hor / lob + 1;

v.pb(my);

Egyptian(lob\*my-hor,my\*hor);

}

LL limit = (LL)1e6;

LL ans[100],s;

LL F(LL lob,LL hor,LL id)

{

if(lob == 0 || hor == 0)return 0;

LL g = \_\_gcd(lob,hor);

lob /= g;

hor /= g;

if(hor > limit)return hor;

if(lob == 1)

{

s = id;

return ans[id] = hor;

}

LL my = hor / lob + 1;

LL ret = (LL)1e15;

for(LL i = my; i <= limit; i++)

{

ret = min( ret , F(lob\*i-hor,i\*hor,id+1) );

if(ret <= limit)

{

ans[id] = i;

break;

}

}

return ret;

}

**Hashing:**

#define M 2

LL base[2] = {29,31};

LL mod[2] = {2091573227,2117566807};

LL p[M][Max],inv[M][Max];

void prework(int N)

{

p[0][0] = inv[0][0] = 1;

p[1][0] = inv[1][0] = 1;

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

{

LL d = bigmod(base[k],mod[k]-2,mod[k]);

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

{

p[k][i] = ( p[k][i-1] \* base[k] ) % mod[k];

inv[k][i] = ( inv[k][i-1] \* d ) % mod[k];

}

}

}

struct Hash

{

char str[Max];

int n;

LL h[M][Max],rv[M][Max];

void init()

{

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

{

LL ret = 0;

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

{

ret = ( ret + p[k][i] \* ( str[i] - 'a' + 1 ) ) % mod[k];

h[k][i] = ret;

}

}

}

void reverse\_init()

{

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

{

LL ret = 0;

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

{

ret = ( ret + p[k][n-i-1] \* ( str[i] - 'a' + 1 ) ) % mod[k];

rv[k][i] = ret;

}

}

}

LL get\_hash(int l,int r)

{

LL ret[M];

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

{

ret[k] = h[k][r];

if(l) ret[k] -= h[k][l-1];

if( ret[k] < 0 ) ret[k] += mod[k];

ret[k] = ( ret[k] \* inv[k][l] ) % mod[k];

}

///return ret[0];

return ret[0] + ret[1] \* 2200000000;

}

LL get\_rev\_hash(int l,int r)

{

LL ret[M];

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

{

ret[k] = rv[k][l];

if( r < (n-1) ) ret[k] -= rv[k][r+1];

if( ret[k] < 0 ) ret[k] += mod[k];

ret[k] = ( ret[k] \* inv[k][n-1-r] ) % mod[k];

}

///return ret[0];

return ret[0] + ret[1] \* 2200000000;

}

bool ispalindrome(int l,int r)

{

return get\_hash(l,r) == get\_rev\_hash(l,r);

}

};

Hash obj;

**2-Edge Split:**

int n,m;

vector <int> g[Max],ans;

bool visit[Max];

void Push(int u,int v,int w) { ans.pb(u); ans.pb(v); ans.pb(w); }

int dfs(int u)

{

visit[u] = 1;

vector <int> keep,baki;

for(int i = 0; i < sz(g[u]); i++)

{

int v = g[u][i];

if( visit[v] == 0 ) keep.pb(v);

}

for(int i = 0; i < sz(keep); i++)

{

int v = keep[i];

int w = dfs(v);

if(w) Push(u,v,w);

else baki.pb(v);

}

int s = ( sz(baki) / 2 ) \* 2;

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

{

int v = baki[i];

int w = baki[i+1];

Push(v,u,w);

}

if( s != sz(baki) ) return baki[s];

return 0;

}

**Russian Peasant:**

ULL RP(ULL a,ULL b)

{

ULL ret = 0;

while (b > 0)

{

if(b & 1)

{

ret += a;

}

if(ret > t)return INF;

a = a << 1;

b = b >> 1;

}

return ret;

}

**Theorem:**

Pick's Theorem

Area = Interior\_Point + Border\_Point / 2 – 1

Dearrangement

D(m,n) = (m-n) \* D(m-1,n-1) + (n-1) \* ( D(m-1,n-1) + D(m-2,n-2) )

#define gamma 0.5772156649

#define harm(x) log(x) + gamma + 1.0/(2\*x) - 1.0/(12\*sqr(x))

#define joshephus(n,k) j(int n, int k) {ll res = 1; for(ll i=2; i<=n; i++) res = (res+k-1) % i + 1; return res;}