Spear of Triam – Team Notebook [ ACM – ICPC ]

**#include** <set>

**#include** <map>

**#include** <list>

**#include** <cmath>

**#include** <queue>

**#include** <stack>

**#include** <cstdio>

**#include** <string>

**#include** <vector>

**#include** <cstdlib>

**#include** <cstring>

**#include** <sstream>

**#include** <iomanip>

**#include** <iostream>

**#include** <algorithm>

**#include** <ctime>

**#include** <deque>

**#include** <bitset>

**#include** <cctype>

**#include** <utility>

**using** **namespace** std;

**typedef** **long** **long** ll;

**typedef** **long double** ld;

**typedef** **unsigned** **int** ui;

**typedef** **unsigned** **long** **long** ull;

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

**#define** Repd(i,n) **for**(**int** i = (n)-1; i >= 0; --i)

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

**#define** Ford(i,a,b) **for**(**int** i = (a); i >= (b); --i)

**#define** Fit(i,v) **for**(**\_\_typeof**((v).begin()) i = (v).begin(); i != (v).end(); ++i)

**#define** Fitd(i,v) **for**(**\_\_typeof**((v).rbegin()) i = (v).rbegin(); i != (v).rend(); ++i)

**#define** mp make\_pair

**#define** pb push\_back

**#define** fi first

**#define** se second

**#define** sz(a) ((**int**)(a).size())

**#define** all(a) (a).begin(), (a).end()

**#define** ms(a,x) memset(a, x, **sizeof**(a))

**template**<**class** **F**, **class** **T**> **T** **convert**(**F** a, **int** p = -1) { stringstream ss; **if** (p >= 0) ss << fixed << setprecision(p); ss << a; **T** r; ss >> r; **return** r; }

**template**<**class** **T**> **T** **gcd**(**T** a, **T** b){ **T** r; **while** (b != 0) { r = a % b; a = b; b = r; } **return** a;}

**template**<**class** **T**> **T** **lcm**(**T** a, **T** b) { **return** a / gcd(a, b) \* b; }

**template**<**class** **T**> **T** **sqr**(**T** x) { **return** x \* x; }

**template**<**class** **T**> **T** **cube**(**T** x) { **return** x \* x \* x; }

**template**<**class** **T**> **int** **getbit**(**T** s, **int** i) { **return** (s >> i) & 1; }

**template**<**class** **T**> **T** **onbit**(**T** s, **int** i) { **return** s | (**T**(1) << i); }

**template**<**class** **T**> **T** **offbit**(**T** s, **int** i) { **return** s & (~(**T**(1) << i)); }

**template**<**class** **T**> **int** **cntbit**(**T** s) { **return** s == 0 ? 0 : cntbit(s >> 1) + (s & 1); }

**typedef** pair<**int**, **int**> II;

**const** **ld** PI = **acos**(-1.0);

**const** **ld** eps = 1e-9;

**const** **int** dr[] = {-1, 0, +1, 0};

**const** **int** dc[] = {0, +1, 0, -1};

**const** **int** inf = (**int**)1e9 + 5;

**const** ll linf = (ll)1e16 + 5;

**const** ll mod = (ll)1e9 + 7;

## Đọc dữ liệu nhanh

**const** **int** bfsz = 1 << 16; **char** bf[bfsz + 5]; **int** rsz = 0;**int** ptr = 0;

**char** **gc**() { **if** (rsz <= 0) { ptr = 0; rsz = (**int**) **fread**(bf, 1, bfsz, stdin); **if** (rsz <= 0) **return** EOF; } --rsz; **return** bf[ptr++]; }

**void** **ga**(**char** &c) { c = EOF; **while** (!**isalpha**(c)) c = gc(); }

**int** **gs**(**char** s[]) { **int** l = 0; **char** c = gc(); **while** (**isspace**(c)) c = gc(); **while** (c != EOF && !**isspace**(c)) { s[l++] = c; c = gc(); } s[l] = '\0'; **return** l; }

**template**<**class** **T**> **bool** **gi**(**T** &v) {

v = 0; **char** c = gc(); **while** (c != EOF && c != '-' && !**isdigit**(c)) c = gc(); **if** (c == EOF) **return** **false**; **bool** neg = c == '-'; **if** (neg) c = gc();

**while** (**isdigit**(c)) { v = v \* 10 + c - '0'; c = gc(); } **if** (neg) v = -v; **return** **true**;

}

**Cin/Cout nhanh** (Không mix với scanf/print, dùng ‘\n’ thay endl)

ios\_base::**sync\_with\_stdio**(**false**);

cin.tie(NULL);

## Luồng cực đại (Dinic’s algorithm)

Các đỉnh đánh số từ 0 -> n

**struct** Dinic {

**int** n, s, t, E, adj[maxe], flow[maxe], cap[maxe], next[maxe], last[maxv], run[maxv], level[maxv], que[maxv];

**void** **init**(**int** \_n, **int** \_s, **int** \_t) {

n = \_n; s = \_s; t = \_t; E = 0;

For(i, 0, n) last[i] = -1;

}

**void** **add**(**int** u, **int** v, **int** c1, **int** c2) {

adj[E] = v; flow[E] = 0; cap[E] = c1; next[E] = last[u]; last[u] = E++;

adj[E] = u; flow[E] = 0; cap[E] = c2; next[E] = last[v]; last[v] = E++;

}

**bool** **bfs**() {

For(i, 0, n) level[i] = -1;

level[s] = 0;

**int** qsize = 0;

que[qsize++] = s;

Rep(i, qsize) {

**for** (**int** u = que[i], e = last[u]; e != -1; e = next[e]) {

**int** v = adj[e];

**if** (flow[e] < cap[e] && level[v] == -1) {

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

que[qsize++] = v;

}

}

}

**return** level[t] != -1;

}

**int** **dfs**(**int** u, **int** bot) {

**if** (u == t) **return** bot;

**for** (**int** &e = run[u]; e != -1; e = next[e]) {

**int** v = adj[e], delta = 0;

**if** (level[v] == level[u] + 1 && flow[e] < cap[e] && (delta = dfs(v, min(bot, cap[e] - flow[e]))) > 0) {

flow[e] += delta; flow[e ^ 1] -= delta;

**return** delta;

}

}

**return** 0;

}

**int** **maxflow**() {

**int** total = 0;

**while** (bfs()) {

For(i, 0, n) run[i] = last[i];

**for** (**int** delta = dfs(s, inf); delta > 0; delta = dfs(s, inf)) total += delta;

}

**return** total;

}

} dinic;

## Luồng cực đại trọng số cực tiểu

**struct** MincostFlow {

**int** n, s, t, E, adj[maxe], next[maxe], last[maxv], which[maxv];

ll totalCost, totalFlow, cap[maxe], flow[maxe], cost[maxe], pot[maxe], dist[maxv];

**void** **init**(**int** \_n, **int** \_s, **int** \_t) {

n = \_n; s = \_s; t = \_t;

ms(last, -1); E = 0;

}

**void** **add**(**int** u, **int** v, ll ca, ll co) {

adj[E] = v; cap[E] = ca; flow[E] = 0; cost[E] = +co; next[E] = last[u]; last[u] = E++;

adj[E] = u; cap[E] = 0; flow[E] = 0; cost[E] = -co; next[E] = last[v]; last[v] = E++;

}

**void** **bellman**() {

**bool** stop = **false**;

ms(pot, 0);

**while** (!stop) {

stop = **true**;

Rep(u, n) **for** (**int** e = last[u]; e != -1; e = next[e]) **if** (flow[e] < cap[e]) {

**int** v = adj[e];

**if** (pot[v] > pot[u] + cost[e]) {

pot[v] = pot[u] + cost[e];

stop = **false**;

}

}

}

}

**bool** **dijkstra**() {

**typedef** pair<ll, **int**> node;

priority\_queue<node, vector<node>, greater<node> > que;

Rep(u, n) dist[u] = linf;

dist[s] = 0;

que.push(mp(0, s));

**while** (!que.empty()) {

ll dnow = que.top().fi;

**int** u = que.top().se;

que.pop();

**if** (dnow > dist[u]) **continue**;

**for** (**int** e = last[u]; e != -1; e = next[e]) **if** (flow[e] < cap[e]) {

**int** v = adj[e];

ll dnext = dnow + cost[e] + pot[u] - pot[v];

**if** (dist[v] > dnext) {

dist[v] = dnext;

which[v] = e;

que.push(mp(dnext, v));

}

}

}

**return** dist[t] < linf;

}

**bool** **maxflow**(ll desireFlow = linf) {

totalCost = totalFlow = 0;

bellman();

**while** (totalFlow < desireFlow) {

**if** (!dijkstra()) **return** **false**;

ll delta = desireFlow - totalFlow;

**for** (**int** v = t, e = which[v]; v != s; v = adj[e ^ 1], e = which[v]) delta = min(delta, cap[e] - flow[e]);

**for** (**int** v = t, e = which[v]; v != s; v = adj[e ^ 1], e = which[v]) {

flow[e] += delta;

flow[e ^ 1] -= delta;

}

totalFlow += delta;

totalCost += delta \* (dist[t] - pot[s] + pot[t]);

Rep(u, n) pot[u] += dist[u];

}

**return** **true**;

}

} mcf;

## Cặp ghép cực đại trên đồ thị tổng quát (Edmonds Lawler algorithm)

Các đỉnh đánh số từ 1 -> n, khi add cạnh phải add cả 2 chiều

**struct** EdmondsLawler {

**int** n, E, start, finish, newRoot, qsize, adj[maxe], next[maxe], last[maxv], mat[maxv], que[maxv], dad[maxv], root[maxv];

**bool** inque[maxv], inpath[maxv], inblossom[maxv];

**void** **init**(**int** \_n) {

n = \_n; E = 0;

For(x, 1, n) { last[x] = -1; mat[x] = 0; }

}

**void** **add**(**int** u, **int** v) {

adj[E] = v; next[E] = last[u]; last[u] = E++;

}

**int** **lca**(**int** u, **int** v) {

For(x, 1, n) inpath[x] = **false**;

**while** (**true**) {

u = root[u];

inpath[u] = **true**;

**if** (u == start) **break**;

u = dad[mat[u]];

}

**while** (**true**) {

v = root[v];

**if** (inpath[v]) **break**;

v = dad[mat[v]];

}

**return** v;

}

**void** **trace**(**int** u) {

**while** (root[u] != newRoot) {

**int** v = mat[u];

inblossom[root[u]] = **true**;

inblossom[root[v]] = **true**;

u = dad[v];

**if** (root[u] != newRoot) dad[u] = v;

}

}

**void** **blossom**(**int** u, **int** v) {

For(x, 1, n) inblossom[x] = **false**;

newRoot = lca(u, v);

trace(u); trace(v);

**if** (root[u] != newRoot) dad[u] = v;

**if** (root[v] != newRoot) dad[v] = u;

For(x, 1, n) **if** (inblossom[root[x]]) {

root[x] = newRoot;

**if** (!inque[x]) {

inque[x] = **true**;

que[qsize++] = x;

}

}

}

**bool** **bfs**() {

For(x, 1, n) {

inque[x] = **false**;

dad[x] = 0;

root[x] = x;

}

qsize = 0;

que[qsize++] = start;

inque[start] = **true**;

finish = 0;

Rep(i, qsize) {

**int** u = que[i];

**for** (**int** e = last[u]; e != -1; e = next[e]) {

**int** v = adj[e];

**if** (root[v] != root[u] && v != mat[u]) {

**if** (v == start || (mat[v] > 0 && dad[mat[v]] > 0)) blossom(u, v);

**else** **if** (dad[v] == 0) {

dad[v] = u;

**if** (mat[v] > 0) que[qsize++] = mat[v];

**else** {

finish = v;

**return** **true**;

}

}

}

}

}

**return** **false**;

}

**void** **enlarge**() {

**int** u = finish;

**while** (u > 0) {

**int** v = dad[u], x = mat[v];

mat[v] = u;

mat[u] = v;

u = x;

}

}

**int** **maxmat**() {

For(x, 1, n) **if** (mat[x] == 0) {

start = x;

**if** (bfs()) enlarge();

}

**int** ret = 0;

For(x, 1, n) **if** (mat[x] > x) ++ret;

**return** ret;

}

} edmonds;

## Cặp ghép cực đại trên đồ thị hai phía (Hopcroft Karp algorithm)

Các đỉnh đánh số từ 1->nx, 1->ny

**struct** HopcroftKarp {

**int** nx, ny, E, adj[maxe], next[maxe], last[maxv], run[maxv], level[maxv], que[maxv], matx[maxv], maty[maxv];

**void** **init**(**int** \_nx, **int** \_ny) {

nx = \_nx; ny = \_ny;

E = 0; ms(last, -1);

ms(matx, -1); ms(maty, -1);

}

**void** **add**(**int** x, **int** y) {

adj[E] = y; next[E] = last[x]; last[x] = E++;

}

**bool** **bfs**() {

**int** qsize = 0;

For(x, 1, nx) **if** (matx[x] != -1) level[x] = -1;

**else** {

level[x] = 0;

que[qsize++] = x;

}

**bool** found = **false**;

Rep(i, qsize) {

**for** (**int** x = que[i], e = last[x]; e != -1; e = next[e]) {

**int** y = adj[e];

**if** (maty[y] == -1) found = **true**;

**else** **if** (level[maty[y]] == -1) {

level[maty[y]] = level[x] + 1;

que[qsize++] = maty[y];

}

}

}

**return** found;

}

**int** **dfs**(**int** x) {

**for** (**int** &e = run[x]; e != -1; e = next[e]) {

**int** y = adj[e];

**if** (maty[y] == -1 || (level[maty[y]] == level[x] + 1 && dfs(maty[y]))) {

matx[x] = y;

maty[y] = x;

**return** 1;

}

}

**return** 0;

}

**int** **maxmat**() {

**int** total = 0;

**while** (bfs()) {

For(x, 1, nx) run[x] = last[x];

For(x, 1, nx) **if** (matx[x] == -1) total += dfs(x);

}

**return** total;

}

} hopkarp;

## Cặp ghép cực đại trọng số cực tiểu trên đồ thị hai phía (Hungary algorithm)

Các đỉnh đánh số từ 1->nx, 1->ny, trọng số cost[x][y] >= 0

maty[y] lưu chỉ số đỉnh x ghép với y

fx[x] + fy[y] <= cost[x][y] với mọi x, y

**struct** Hungary {

**int** nx, ny, cost[maxn][maxn], fx[maxn], fy[maxn], maty[maxn], which[maxn], dist[maxn];

**bool** used[maxn];

**void** **init**(**int** \_nx, **int** \_ny) {

nx = \_nx; ny = \_ny; ms(fx, 0); ms(fy, 0);

For(i, 0, nx) For(j, 0, ny) cost[i][j] = inf;

}

**void** **add**(**int** x, **int** y, **int** c) { cost[x][y] = c; }

**int** **mincost**() {

For(x, 1, nx) {

**int** y0 = 0; maty[0] = x;

For(y, 0, ny) { dist[y] = inf + 1; used[y] = **false**; }

**do** {

used[y0] = **true**;

**int** x0 = maty[y0], delta = inf + 1, y1;

For(y, 1, ny) **if** (!used[y]) {

**int** curdist = cost[x0][y] - fx[x0] - fy[y];

**if** (curdist < dist[y]) {

dist[y] = curdist;

which[y] = y0;

}

**if** (dist[y] < delta) {

delta = dist[y];

y1 = y;

}

}

For(y, 0, ny) **if** (used[y]) {

fx[maty[y]] += delta;

fy[y] -= delta;

} **else** dist[y] -= delta;

y0 = y1;

} **while** (maty[y0] != 0);

**do** {

**int** y1 = which[y0];

maty[y0] = maty[y1];

y0 = y1;

} **while** (y0);

}

// return -fy[0]; // nếu luôn đảm bảo có bộ ghép đầy đủ

**int** ret = 0;

For(y, 1, ny) {

**int** x = maty[y];

**if** (cost[x][y] < inf) ret += cost[x][y];

}

**return** ret;

}

## } hungary;

## Cây khung nhỏ nhất trên đồ thị có hướng (Chu-Liu algorithm)

Các đỉnh đánh số từ 0 -> n-1

**namespace** chuliu {

**struct** Cost;

vector<Cost> costlist;

**struct** Cost {

**int** id, val, used, a, b, pos;

**Cost**() { val = -1; used = 0; }

**Cost**(**int** \_id, **int** \_val, **bool** temp) {

a = b = -1; id = \_id; val = \_val; used = 0;

pos = sz(costlist); costlist.pb(\***this**);

}

**Cost**(**int** \_a, **int** \_b) {

a = \_a; b = \_b; id = -1; val = costlist[a].val - costlist[b].val;

used = 0; pos = sz(costlist); costlist.pb(\***this**);

}

**void** **push**() {

**if** (id == -1) {

costlist[a].used += used;

costlist[b].used -= used;

}

}

};

**struct** Edge {

**int** u, v;

Cost cost;

**Edge**() {}

**Edge**(**int** id, **int** \_u, **int** \_v, **int** c) {

u = \_u; v = \_v; cost = Cost(id, c, 0);

}

} edge[maxe];

**int** n, m, root, pre[maxv], node[maxv], vis[maxv], best[maxv];

**void** **init**(**int** \_n) {

n = \_n; m = 0;

costlist.clear();

}

**void** **add**(**int** id, **int** u, **int** v, **int** c) {

edge[m++] = Edge(id, u, v, c);

}

**int** **mst**(**int** root) {

**int** ret = 0;

**while** (**true**) {

Rep(i, n) best[i] = -1;

Rep(e, m) {

**int** u = edge[e].u, v = edge[e].v;

**if** ((best[v] == -1 || edge[e].cost.val < costlist[best[v]].val) && u != v) {

pre[v] = u;

best[v] = edge[e].cost.pos;

}

}

Rep(i, n) **if** (i != root && best[i] == -1) **return** -1;

**int** cntnode = 0;

ms(node, -1); ms(vis, -1);

Rep(i, n) **if** (i != root) {

ret += costlist[best[i]].val;

costlist[best[i]].used++;

**int** v = i;

**while** (vis[v] != i && node[v] == -1 && v != root) {

vis[v] = i;

v = pre[v];

}

**if** (v != root && node[v] == -1) {

**for** (**int** u = pre[v]; u != v; u = pre[u]) node[u] = cntnode;

node[v] = cntnode++;

}

}

**if** (cntnode == 0) **break**;

Rep(i, n) **if** (node[i] == -1) node[i] = cntnode++;

Rep(e, m) {

**int** v = edge[e].v;

edge[e].u = node[edge[e].u];

edge[e].v = node[edge[e].v];

**if** (edge[e].u != edge[e].v) edge[e].cost = Cost(edge[e].cost.pos, best[v]);

}

n = cntnode;

root = node[root];

}

**return** ret;

}

vector<**int**> **trace**() {

vector<**int**> ret;

Repd(i, sz(costlist)) costlist[i].push();

Rep(i, sz(costlist)) {

Cost cost = costlist[i];

**if** (cost.id != -1 && cost.used > 0) ret.pb(cost.id);

}

**return** ret;

}

}

## Stable marriage problem

**Ghép n nam với n nữ sao cho không tồn tại 2 cặp (i, j) và (x, y) nào mà:**

**i thích y hơn j, và y cũng thích i hơn x.**

**int** n, score[maxn][maxn], husband[maxn];

queue<**int**> womanlist[maxn];

queue<**int**> freemans;

**bool** **prefer**(**int** woman, **int** newman, **int** herhusband) {

**return** score[woman][newman] > score[woman][herhusband];

}

**void** **doit**() {

ms(husband, -1);

For(man, 1, n) freemans.push(man);

**while** (!freemans.empty()) {

**int** man = freemans.front();

**int** woman = womanlist[man].front();

womanlist[man].pop();

**if** (husband[woman] == -1) {

husband[woman] = man;

freemans.pop();

} **else** {

**int** herhusband = husband[woman];

**if** (prefer(woman, man, herhusband)) {

husband[woman] = man;

freemans.pop();

freemans.push(herhusband);

}

}

}

For(woman, n + 1, n + n) **printf**("(%d %d) ", husband[woman], woman);

}

**int** **main**() {

gi(n);

For(man, 1, n) {

**int** woman;

For(i, 1, n) {

gi(woman);

womanlist[man].push(woman);

}

}

For(woman, n + 1, n + n) {

**int** man;

For(i, 1, n) {

gi(man);

score[woman][man] = n - i + 1;

}

}

doit();

}

## 2-SAT

Các đỉnh trong đồ thị ban đầu đánh số từ 0 -> n-1

Các đỉnh trong đồ thị 2-sat đánh số từ 0 -> 2\*n-1

Trong đó u và !u là 2 đỉnh liên tiếp [0-1], [2-3], [4-5], …

Lưu ý đặt maxv = 2\*maxn

**#define** maxv 20000

**#define** pos(v) ((v) << 1)

**#define** neg(v) (pos(v) ^ 1)

**namespace** twosat {

**int** n;

vector<**int**> forward[maxv], backward[maxv];

**bool** used[maxv];

**int** cnt, order[maxv], comp[maxv];

**void** **init**(**int** \_n) {

n = \_n;

Rep(i, n) {

forward[i].clear();

backward[i].clear();

}

}

**void** **add**(**int** u, **int** v) {

forward[u].pb(v);

backward[v].pb(u);

}

**void** **dfs1**(**int** u) {

used[u] = **true**;

Rep(i, sz(forward[u])) {

**int** v = forward[u][i];

**if** (!used[v]) dfs1(v);

}

order[cnt++] = u;

}

**void** **dfs2**(**int** u, **int** c) {

comp[u] = c;

Rep(i, sz(backward[u])) {

**int** v = backward[u][i];

**if** (comp[v] == -1) dfs2(v, c);

}

}

**bool** **solve**(vector<**int**> &ret) {

cnt = 0;

ms(used, **false**);

Rep(u, n) **if** (!used[u]) dfs1(u);

ms(comp, -1);

**int** c = 0;

Repd(i, n) {

**int** u = order[i];

**if** (comp[u] == -1) dfs2(u, c++);

}

**for** (**int** u = 0; u < n; u += 2)

**if** (comp[u] == comp[u ^ 1]) **return** **false**;

ret.clear();

**for** (**int** u = 0; u < n; u += 2) {

**int** choose = (comp[u] > comp[u ^ 1]) ? u : u ^ 1;

**if** (!(choose & 1)) ret.pb(choose >> 1);

}

**return** **true**;

}

}

## Lyndon decomposition

Phân tích xâu s = w1w2..wk mà w1 ≥ w2 ≥ .. ≥ wk

**void** **lyndon**(string s) {

**int** n = (**int**) s.length();

**int** i = 0;

**while** (i < n) {

**int** j = i + 1, k = i;

**while** (j < n && s[k] <= s[j]) {

**if** (s[k] < s[j]) k = i;

**else** ++k;

++j;

}

**while** (i <= k) {

cout << s.substr(i, j - k) << ' ';

i += j - k;

}

}

}

## Minimum rotation

**Tính vị trí của xâu xoay vòng có thứ tự từ điển nhỏ nhất của xâu s[]**

**int** **minmove**(**char** s[], **int** n) {

**int** x, y, i, j, u, v; // x is the smallest string before string y

**for** (x = 0, y = 1; y < n; ++ y) {

i = u = x;

j = v = y;

**while** (s[i] == s[j]) {

++ u; ++ v;

**if** (++ i == n) i = 0;

**if** (++ j == n) j = 0;

**if** (i == x) **break**; // All strings are equal

}

**if** (s[i] <= s[j]) y = v;

**else** {

x = y;

**if** (u > y) y = u;

}

}

**return** x;

}

## Manacher’s algorithm

**d[i] = bán kính của Palindrome dài nhất có tâm tại ví trí i**

**int** **manacher**(char s[], int n) {

**int** res = 0;

// Palindrom độ dài lẻ

**int** l = 0, r = -1;

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

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

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

d[i] = --k;

res = max(res, k + k + 1);

**if** (r < i + k) {

l = i - k;

r = i + k;

}

}

// Palindrome độ dài chẵn

l = 0; r = -1;

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

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

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

d[i] = --k;

res = max(res, k + k);

**if** (r < i + k - 1) {

l = i - k;

r = i + k - 1;

}

}

**return** res; // Độ dài palindrome dài nhất là xâu con của s[]

}

## Z-Function

**z[i] = độ dài xâu con dài nhất bắt đầu từ vị trí i mà trùng với đoạn đầu của v[]**

**void** **zfunc**(**int** n, **int** v[], **int** z[]) {

**int** l = 0, r = -1;

z[0] = 0;

For(i, 1, n - 1) {

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

**while** (i + k < n && v[k] == v[i + k]) ++k;

z[i] = k;

**if** (i + k - 1 > r) {

l = i;

r = i + k - 1;

}

}

z[0] = n;

}

## Maine-Lorentz Algorithm

Tìm các xâu con lặp liên tiếp 2 lần

vector<**int**> **z\_function**(**const** string & s) {

**int** n = (**int**) s.length();

vector<**int**> z(n);

**for** (**int** i = 1, l = 0, r = 0; i < n; ++i) {

**if** (i <= r)

z[i] = min(r - i + 1, z[i - l]);

**while** (i + z[i] < n && s[z[i]] == s[i + z[i]])

++z[i];

**if** (i + z[i] - 1 > r)

l = i, r = i + z[i] - 1;

}

**return** z;

}

**void** **output\_tandem**(**const** string & s, **int** shift, **bool** left, **int** cntr, **int** l,

**int** l1, **int** l2) {

**int** pos;

**if** (left) pos = cntr - l1;

**else** pos = cntr - l1 - l2 - l1 + 1;

cout << "[" << shift + pos << ".." << shift + pos + 2 \* l - 1 << "] = "

<< s.substr(pos, 2 \* l) << **endl**;

}

**void** **output\_tandems**(**const** string & s, **int** shift, **bool** left, **int** cntr, **int** l,

**int** k1, **int** k2) {

**for** (**int** l1 = 1; l1 <= l; ++l1) {

**if** (left && l1 == l)

**break**;

**if** (l1 <= k1 && l - l1 <= k2)

output\_tandem(s, shift, left, cntr, l, l1, l - l1);

}

}

**inline** **int** **get\_z**(**const** vector<**int**> & z, **int** i) {

**return** 0 <= i && i < (**int**) z.size() ? z[i] : 0;

}

**void** **find\_tandems**(string s, **int** shift = 0) {

**int** n = (**int**) s.length();

**if** (n == 1)

**return**;

**int** nu = n / 2, nv = n - nu;

string u = s.substr(0, nu), v = s.substr(nu);

string ru = string(u.rbegin(), u.rend()), rv = string(v.rbegin(), v.rend());

find\_tandems(u, shift);

find\_tandems(v, shift + nu);

vector<**int**> z1 = z\_function(ru), z2 = z\_function(v + '#' + u), z3 =

z\_function(ru + '#' + rv), z4 = z\_function(v);

**for** (**int** cntr = 0; cntr < n; ++cntr) {

**int** l, k1, k2;

**if** (cntr < nu) {

l = nu - cntr;

k1 = get\_z(z1, nu - cntr);

k2 = get\_z(z2, nv + 1 + cntr);

} **else** {

l = cntr - nu + 1;

k1 = get\_z(z3, nu + 1 + nv - 1 - (cntr - nu));

k2 = get\_z(z4, (cntr - nu) + 1);

}

**if** (k1 + k2 >= l) output\_tandems(s, shift, cntr < nu, cntr, l, k1, k2);

}

}

## Aho-corasick

**int** key[maxnode], next[maxnode], ptrlist, nil;

**struct** List {

**int** head, tail;

**void** **init**() { head = tail = nil; }

**void** **add**(**int** k) {

++ptrlist;

key[ptrlist] = k;

next[ptrlist] = nil;

**if** (head == nil) {

head = tail = ptrlist;

**return**;

}

next[tail] = ptrlist;

tail = ptrlist;

}

**void** **join**(List &other) {

**if** (other.head == nil) **return**;

**if** (head == nil) {

head = other.head;

tail = other.tail;

**return**;

}

next[tail] = other.head;

tail = other.tail;

}

};

**int** go[maxnode][maxc], fail[maxnode], ptrTrie, que[maxnode], qsize;

List out[maxnode];

**void** **init**() {

nil = ptrlist = -1;

ptrTrie = 0;

ms(go[0], 0);

out[0].init();

}

**void** **add**(**int** key, **char** s[], **int** len) {

**int** v = 0;

Rep(i, len) {

**int** c = s[i] - 'a';

**if** (go[v][c] < 1) {

++ptrTrie;

ms(go[ptrTrie], -1);

out[ptrTrie].init();

go[v][c] = ptrTrie;

}

v = go[v][c];

}

out[v].add(key);

}

**void** **build**() {

qsize = 0;

Rep(c, maxc) **if** (go[0][c]) {

fail[go[0][c]] = 0;

que[qsize++] = go[0][c];

}

Rep(i, qsize) {

**int** r = que[i];

Rep(c, maxc) {

**int** u = go[r][c];

**if** (u < 0) go[r][c] = go[fail[r]][c];

**else** {

que[qsize++] = u;

fail[u] = go[fail[r]][c];

out[u].join(out[fail[u]]);

}

}

}

}

**void** **search**(**int** s[], **int** ns) {

**int** v = 0;

For(i, 1, ns) {

**int** c = s[i];

**while** (go[v][c] < 0) v = fail[v];

v = go[v][c];

**int** x = out[v].head;

**while** (x != nil) {

**int** k = key[x];

// Found a matching

x = next[x];

}

}

}

## Suffix array

**int** n, from, to, nc, a[maxn], p[maxn], c[maxn], pn[maxn], cn[maxn], cnt[maxn], lcp[maxn];

**char** s[maxn];

**int** **suffix\_array**() {

Rep(i, n) a[i] = s[i] - 'a' + 1;

a[n++] = 0;

ms(cnt, 0);

Rep(i, n) ++cnt[a[i]];

For(i, 1, maxa - 1) cnt[i] += cnt[i - 1];

Repd(i, n) p[--cnt[a[i]]] = i;

nc = 1; c[p[0]] = 0;

For(i, 1, n - 1) {

**if** (a[p[i]] > a[p[i - 1]]) ++nc;

c[p[i]] = nc - 1;

}

**for** (**int** h = 1; h < n; h <<= 1) {

Rep(i, n) {

pn[i] = p[i] - h;

**if** (pn[i] < 0) pn[i] += n;

}

ms(cnt, 0);

Rep(i, n) ++cnt[c[pn[i]]];

For(i, 1, nc - 1) cnt[i] += cnt[i - 1];

Repd(i, n) p[--cnt[c[pn[i]]]] = pn[i];

nc = 1; cn[p[0]] = 0;

For(i, 1, n - 1) {

**int** x = (p[i] + h) % n, y = (p[i - 1] + h) % n;

**if** (c[p[i]] > c[p[i - 1]] || c[x] > c[y]) ++nc;

cn[p[i]] = nc - 1;

}

Rep(i, n) c[i] = cn[i];

}

**int** h = 0;

Rep(i, n) {

**if** (c[i] > 0) {

**int** j = p[c[i] - 1];

**while** (a[i + h] == a[j + h]) ++h;

lcp[c[i]] = h;

**if** (h > 0) --h;

}

}

**int** ret = 0; // Tính số xâu con phân biệt độ dài trong khoảng [from, to]

For(i, 1, n - 1) {

**int** x = max(lcp[i] + 1, from);

**int** y = min(n - 1 - p[i], to);

**if** (x <= y) ret += y - x + 1;

}

**return** ret;

}

## Nim Product

**const** **int** MAXB = 10;

**struct** NimProduct {

**int** bit[MAXB + 1][MAXB + 1];

**NimProduct**() { ms(bit, -1); }

// Compute Nim Multiplication: 2^x \* 2^y

**int** **nimbit**(**int** x, **int** y) {

**if** (!(x & y)) **return** 1 << (x | y);

**int** &ret = bit[x][y];

**if** (ret != -1) **return** ret;

**if** (x > y) **return** ret = nimbit(y, x);

ret = 1;

**for** (**int** i = 1; i <= y; i <<= 1)

**if** (x & y & i) ret = nimmul(ret, (1 << i) \* 3 >> 1);

**else** **if** ((x | y) & i) ret = nimmul(ret, 1 << i);

**return** ret;

}

// Compute Nim Product x \* y

**int** **nimmul**(**int** x, **int** y)

{

**int** ret = 0;

**for** (**int** bx = 0; bx <= MAXB; ++bx)

**for** (**int** by = 0; by <= MAXB; ++by)

**if** ((1 & (x >> bx)) && (1 & (y >> by)))

ret ^= nimbit(bx, by);

**return** ret;

}

};

## Xoay Rubik

Đếm số lần lặp lại chuỗi thao tác xoay rubik để nó quay trở lại trạng thái ban đầu

U (trên), D (dưới), L (trái), R (phải), F (trước), B (sau)

Chữ thường là xoay 90 độ theo chiều kim đồng hồ, chữ hoa là xoay 90 độ ngược chiều kim đồng hồ

**int** r[][8] = { { 1, 2, 3, 6, 9, 8, 7, 4 }, { 28, 29, 30, 33, 36, 35, 34, 31 },

{ 37, 38, 39, 42, 45, 44, 43, 40 }, { 19, 20, 21, 24, 27, 26, 25, 22 },

{ 46, 47, 48, 51, 54, 53, 52, 49 }, { 10, 11, 12, 15, 18, 17, 16, 13 } };

**int** c[][12] = { { 21, 20, 19, 10, 13, 16, 43, 44, 45, 54, 51, 48 },

{ 25, 26, 27, 46, 49, 52, 39, 38, 37, 18, 15, 12 },

{ 34, 35, 36, 52, 53, 54, 7, 8, 9, 16, 17, 18 },

{ 3, 2, 1, 48, 47, 46, 30, 29, 28, 12, 11, 10 },

{ 27, 24, 21, 1, 4, 7, 45, 42, 39, 36, 33, 30 },

{ 19, 22, 25, 28, 31, 34, 37, 40, 43, 9, 6, 3 } };

**int** a[55], flag[55], b[55];

**void** **duyet**(**int** x) {

For(i, 1, 54) b[i] = a[i];

Rep(i, 8) b[r[x][i]] = a[r[x][(i + 6) % 8]];

Rep(i, 12) b[c[x][i]] = a[c[x][(i + 9) % 12]];

For(i, 1, 54) a[i] = b[i];

}

ll **rubik**(string query) {

string s = "";

Rep(i, sz(query)) {

**if** (query[i] >= 'a' && query[i] <= 'z') {

query[i] += 'A' - 'a';

s.pb(query[i]); s.pb(query[i]); s.pb(query[i]);

} **else** s.push\_back(query[i]);

}

For(i, 1, 54) a[i] = i;

ms(flag, 0);

Rep(i, sz(s)) {

**if** (s[i] == 'U') duyet(0);

**else** **if** (s[i] == 'D') duyet(1);

**else** **if** (s[i] == 'L') duyet(2);

**else** **if** (s[i] == 'R') duyet(3);

**else** **if** (s[i] == 'F') duyet(4);

**else** duyet(5);

}

ll ret = 1;

For(i, 1, 54) **if** (!flag[i]) {

**int** t = i;

ll nhan = 1;

flag[i] = 1;

**while** (flag[a[t]] == 0) {

t = a[t];

flag[t] = 1;

nhan++;

}

ret = lcm(ret, nhan);

}

**return** ret - 1;

}

## Dice Structure

**struct** die;

map<**int**, die> dieMap;

**struct** die {

**int** arr[6]; /\* 0 right, 1 left, 2 forward, 3 backward, 4 top, 5 bottom \*/

**die**(){ Rep(i,6) arr[i] = i; }

**die**(**int** cipher) { // 0 -> 23

**if** (dieMap.empty()) **puts**("Call openDie(die());"); **else** (\***this**) = dieMap[cipher];

}

die **moveLeft**() {

die res =(\***this**);

**int** t = res.arr[1];

res.arr[1] = res.arr[4]; res.arr[4] = res.arr[0]; res.arr[0] = res.arr[5]; res.arr[5] = t;

**return** res;

}

die **moveRight**() {

die res =(\***this**);

**int** t = res.arr[1];

res.arr[1] = res.arr[5]; res.arr[5] = res.arr[0]; res.arr[0] = res.arr[4]; res.arr[4] = t;

**return** res;

}

die **moveUp**() {

die res =(\***this**);

**int** t = res.arr[4];

res.arr[4] = res.arr[3]; res.arr[3] = res.arr[5]; res.arr[5] = res.arr[2]; res.arr[2] = t;

**return** res;

}

die **moveDown**() {

die res =(\***this**);

**int** t = res.arr[4];

res.arr[4] = res.arr[2]; res.arr[2] = res.arr[5]; res.arr[5] = res.arr[3];

res.arr[3] = t;

**return** res;

}

**int** **encrypt**() { // 0 -> 23

**int** res = arr[0] \* 4;

For(i,3,5) **if** (arr[i] < arr[2]) res++;

**return** res;

}

};

**void** **openDie**(die t) {

dieMap[t.encrypt()] = t;

**if**(dieMap.find(t.moveLeft().encrypt()) == dieMap.end()) openDie(t.moveLeft());

**if**(dieMap.find(t.moveRight().encrypt()) == dieMap.end()) openDie(t.moveRight());

**if**(dieMap.find(t.moveUp().encrypt()) == dieMap.end()) openDie(t.moveUp());

**if**(dieMap.find(t.moveDown().encrypt()) == dieMap.end()) openDie(t.moveDown());

}

## Extended Euclidean

Dùng Extended Euclid để tìm nghiệm của phương trình ax + by = gcd(a, b). Giả sử kết quả trả về là (x0, y0), họ nghiệm của phương trình sẽ là với . Phương trình tổng quát ax + by = d chỉ có nghiệm khi d chia hết cho gcd(a, b).

## Chinese Remainder Theorem

**Tìm nghiệm nhỏ nhất của hệ phương trình đồng dư**

// a, b không âm

pair <ll, ll> **euclid**(ll a, ll b) {

**if** (b == 0) **return** make\_pair(1, 0);

pair <ll, ll> r = euclid(b, a % b);

**return** make\_pair(r.second, r.first - a / b \* r.second);

}

**int** n;

ll p[maxn + 1], r[maxn + 1], b[maxn + 1], x[maxn + 1];

// Trả về -1 nếu vô nghiệm

ll **chineseRemainder**(**int** n, **int** p[], **int** r[]) {

ll M = 1;

**for** (**int** i = 0; i < n; ++i) M \*= p[i];

ll N = 0;

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

b[i] = M / p[i];

x[i] = euclid(b[i], p[i]).first;

N += r[i] \* b[i] \* x[i];

}

**while** (N - M >= 0) N -= M;

**while** (N < 0) N += M;

**return** N;

}

## Congruence Equation

Giải phương trình: a1x1 + a2x2 + … + anxn ≡ b (modul m)

Trong đó a1, a2, …, an, b, m là các số nguyên dương.

**int** g[MAXN], x[MAXN];

**bool** **congruenceEquation**(vector<**int**> a, **int** b, **int** m, vector<**int**> &ret) {

**int** n = sz(a);

a.pb(m);

g[0] = a[0];

For(i, 1, n) g[i] = gcd(g[i - 1], a[i]);

ret.clear();

**if** (b % g[n]) **return** **false**;

**int** val = b / g[n];

Ford(i, n, 1) {

pair<ll, ll> p = extendedEuclid(g[i - 1], a[i]);

x[i] = p.se \* val % m;

val = p.fi \* val % m;

}

x[0] = val;

For(i, 0, n) x[i] = (x[i] + m) % m;

Rep(i, n) ret.pb(x[i]);

**return** **true**;

}

## Rabin Miller

**bool** **rabin**(**long** **long** n) {

**if** (n == 2) **return** **true**;

**if** (n < 2 || !(n & 1)) **return** **false**;

**const** **long** **long** p[3] = {3, 5, 7};

**long** **long** a, d = n-1, mx = 3;

**int** i, r, s = 0;

**while** (!(d & 1)) { ++s; d >>= 1; }

**for** (i = 0; i < mx; ++i) {

**if** (n == p[i]) **return** **true**;

**if** (!(n % p[i])) **return** **false**;

a = pow(p[i], d, n);

**if** (a != 1) {

**for** (r = 0; r < s && a != n-1; ++r) a = mul(a, a, n);

**if** (r == s) **return** **false**;

}

}

**return** **true**;

}

## Tìm một ước không tầm thường của n

**long** **long** **brent**(**long** **long** n) {

**if** (!(n & 1)) **return** 2;

**if** (!(n % 3)) **return** 3;

**const** **int** p[3] = {1, 3, 5};

**long** **long** y, q, x, ys, g, my = 3;

**int** i, j, k, m, r, c;

**for** (i = 0; i < my; ++i) {

y = 1; r = 1; q = 1; m = 111; c = p[i];

**do** {

x = y; k = 0;

**for** (j = 1; j <= r; ++j) y = (mul(y, y, n) + c) % n;

**do** {

ys = y;

**for** (j = 1; j <= min(m, r-k); ++j) {

y = (mul(y, y, n) + c) % n;

q = mul(q, abs(x - y), n);

}

g = gcd(q, n); k += m;

} **while** (k < r && g < 2);

r <<= 1;

} **while** (g < 2);

**if** (g == n)

**do** {

ys = (mul(ys, ys, n) + c) % n;

g = gcd(abs(x - ys), n);

} **while** (g < 2);

**if** (g != n) **return** g;

}

**return** n;

}

## GCD Sum

**Tính tổng ước chung lớn nhất của tất cả các cặp (i, j) với i < j và i, j nằm trong khoảng từ 1 đến N**

**const** **int** N = 1000000;

**int** phi[N + 1];

**int** prime[N + 1];

**long** **long** G[N + 1];

**int** **main**() {

// Compute phi[n]

**for** (**int** p = 1; p <= N; ++ p) phi[p] = p;

**for** (**int** p = 2; p <= N; ++ p)

**if** (phi[p] == p)

**for** (**int** n = p; n <= N; n += p)

phi[n] = phi[n]/p\*(p-1);

// Compute prime[n] -> Largest prime that divides n

**for** (**int** p = 1; p <= N; ++ p) prime[p] = 0;

**for** (**int** p = 2; p <= N; ++ p)

**if** (prime[p] == 0)

**for** (**int** n = p; n <= N; n += p) prime[n] = max(prime[n], p);

G[1] = 1;

**for** (**int** n = 2; n <= N; ++ n) {

**int** p = 1;

**int** k = 0;

**while** (p <= n/prime[n] && n % (p \* prime[n]) == 0) {

p \*= prime[n];

++ k;

}

G[n] = ((k+1)\*p - k\*p/prime[n]) \* G[n/p];

}

G[1] = G[1] - 1;

**for** (**int** n = 2; n <= N; ++ n) G[n] += G[n-1] - n;

}

## LCM Sum

**Tính tổng bội chung nhỏ nhất của tất cả các cặp (i, j) với i < j và i, j nằm trong khoảng từ 1 đến N**

**const** **int** N = 3000000;

**int** prime[N + 1];

ull G[N + 1], F[N + 1];

**int** **main**() {

**for** (**int** n = 1; n <= N; ++n) prime[n] = 0;

**int** sqrtN = (**int**) sqrt(N);

**for** (**int** p = 2; p <= sqrtN; ++p)

**if** (prime[p] == 0)

**for** (**int** n = p \* p; n <= N; n += p) prime[n] = p;

// G[n] = sum(phi[d] \* d) for all d | n ( 1 <= d <= n )

G[1] = 1;

**for** (**int** n = 2; n <= N; ++n) {

**if** (prime[n] == 0) G[n] = (ull) n \* (n - 1) + 1;

**else** {

ull p = prime[n];

ull s = 1; // s = G(p^k) = phi[1] \* 1 + phi[p] \* p + phi[p^2] \* p^2 + ... + phi[p^k] \* p^k

**while** (n % p == 0) {

s += p \* (p - p/prime[n]);

p \*= prime[n];

}

p /= prime[n];

G[n] = s \* G[n/p];

}

}

// F[n] = sum(lcm(x, n)) for all x = 1, 2, 3, .. n - 1

F[1] = 0;

**for** (**int** n = 2; n <= N; ++n) F[n] = (G[n] - 1) / 2 \* n;

// F[n] = sum(lcm(i, j)) for all 1 <= i < j <= n

**for** (**int** n = 2; n <= N; ++n) F[n] += F[n - 1];

}

## Mersenne primes

**const** **int** mersenne[] = { 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521,

607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937,

21701, 23209, 44497, 86243, 110503, 132049, 216091, 756839, 859433,

1257787, 1398269, 2976221, 3021377, 6972593, 13466917, 20996011,

24036583, 25964951, 30402457, 32582657, 37156667, 42643801, 43112609 };

## Đếm số nghiệm nguyên của phương trình Ax + By + C = 0

Với điều kiện: x1 <= x <= x2 và y1 <= y <= y2

Triple **euclid**(ll a, ll b) {

**if** (b == 0) **return** Triple(1, 0, a);

Triple T = euclid(b, a % b);

**return** Triple(T.y, T.x - a / b \* T.y, T.d);

}

ll A, B, C, X1, X2, Y1, Y2;

**int** **sign**(ll a) {

**if** (a == 0) **return** 0;

**return** a < 0 ? -1 : +1;

}

ll **up**(ll a, ll b) {

**if** (a % b == 0) **return** a / b;

**if** (sign(a) \* sign(b) < 0) **return** a / b;

**return** a / b + 1;

}

ll **down**(ll a, ll b) {

**if** (a % b == 0) **return** a / b;

**if** (sign(a) \* sign(b) < 0) **return** a / b - 1;

**return** a / b;

}

ll **howmany**() {

**if** (A < 0) { A = -A; ll X = X1; X1 = -X2; X2 = -X; }

**if** (B < 0) { B = -B; ll Y = Y1; Y1 = -Y2; Y2 = -Y; }

C = -C;

**if** (A == 0 && B == 0) {

**return** C == 0 ? (X2 - X1 + 1) \* (Y2 - Y1 + 1) : 0;

}

**if** (A == 0) {

**if** (C % B != 0) **return** 0;

ll Y = C / B;

**return** Y1 <= Y && Y <= Y2 ? X2 - X1 + 1 : 0;

}

**if** (B == 0) {

**if** (C % A != 0) **return** 0;

ll X = C / A;

**return** X1 <= X && X <= X2 ? Y2 - Y1 + 1 : 0;

}

Triple T = euclid(A, B);

ll X0 = T.x, Y0 = T.y, D = T.d;

**if** (C % D != 0) **return** 0;

ll alpha = A / D, beta = B / D, gama = C / D;

X0 \*= gama; Y0 \*= gama;

ll K1 = -oo, K2 = +oo;

**if** (alpha > 0) {

K1 = max(K1, up(Y0 - Y2, alpha));

K2 = min(K2, down(Y0 - Y1, alpha));

}

**else** {

K1 = max(K1, up(Y0 - Y1, alpha));

K2 = min(K2, down(Y0 - Y2, alpha));

}

**if** (beta > 0) {

K1 = max(K1, up(X1 - X0, beta));

K2 = min(K2, down(X2 - X0, beta));

}

**else** {

K1 = max(K1, up(X2 - X0, beta));

K2 = min(K2, down(X1 - X0, beta));

}

**return** K1 <= K2 ? K2 - K1 + 1 : 0;

}

## Fast Fourier Transform (FFT)

**typedef** complex<ld> cplex;

**int** rev[maxn];

cplex wlen\_pw[maxn], fa[maxn], fb[maxn];;

**void** **fft**(cplex a[], **int** n, **bool** invert) {

**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) {

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

**int** len2 = len >> 1;

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

cplex wlen(cos(alpha), sin(alpha));

**for** (**int** i = 1; i < len2; ++i) wlen\_pw[i] = wlen\_pw[i-1] \* wlen;

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

cplex 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) Rep(i, n) a[i] /= n;

}

**void** **calcRev**(**int** n, **int** logn) {

Rep(i, n) {

rev[i] = 0;

Rep(j, logn) **if** (i & (1 << j)) rev[i] |= 1 << (logn - 1 - j);

}

}

**void** **mulpoly**(**int** a[], **int** b[], ll c[], **int** na, **int** nb, **int** &n) {

**int** l = max(na, nb), logn = 0;

**for** (n = 1; n < l; n <<= 1) ++logn;

n <<= 1; ++logn;

calcRev(n, logn);

Rep(i, n) fa[i] = fb[i] = cplex(0);

Rep(i, na) fa[i] = cplex(a[i]);

Rep(i, nb) fb[i] = cplex(b[i]);

fft(fa, n, **false**);

fft(fb, n, **false**);

Rep(i, n) fa[i] \*= fb[i];

fft(fa, n, **true**);

Rep(i, n) c[i] = (ll)(fa[i].real() + 0.5);

}

## Knight Distance

Tính khoảng cách nhỏ nhất giữa 2 ô theo bước đi của quân mã.

ll **knightDistance**(ll x1, ll y1, ll x2, ll y2) {

ll dx = abs(x2 - x1), dy = abs(y2 - y1);

**if** (dx + dy == 1) **return** 3;

**if** (dx == 2 && dy == 2) **return** 4;

ll ret = max((dx + 1) / 2, (dy + 1) / 2);

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

ret += (ret % 2 != (dx + dy) % 2);

**return** ret;

}

## Degree Theorem:

Dãy không tăng d[] là dãy bậc của đồ thị vô hướng khi và chỉ khi tổng của dãy là chẵn và thỏa mãn điều kiện:

## Pick’s Theorem:

Quan hệ giữa diện tích (A) của một đa giác có các đỉnh nằm trên lưới ô vuông với số điểm nằm trong (i), và số điểm nằm trên cạnh (b) của nó:

## Necklace

Đếm số lượng chuỗi vòng có n hạt, trong đó mỗi hạt có thể được tô bằng 1 trong số k màu. Khi so sánh hai chuỗi hạt, chỉ được phép xoay, không được phép lật:

## Count Graph

Number of labeled graphs with nвершинами равно: vertices is equal to:

Number of non-connected graphs with nвершинами равно: vertices is equal to:

NНаконец, искомое количество связных графов равно:Number of connected graphs is:

Number of labeled graphs with nвершинами и vertices and kкомпонентами связности. connected components.

## Fibonacci

ll **fib**(**int** n) {

**if** (n <= 2) **return** (n ? 1 : 0);

**if** (n & 1) **return** sqr(fib(n >> 1)) + sqr(fib((n >> 1) + 1));

ll x = fib(n >> 1);

**return** (2 \* fib((n >> 1) - 1) + x) \* x;

}

## Josephus

O(m log n)

ll **josephus**(ll m, ll n) {

ll p = m \* n;

**while** (p > n) p = p + (p - n - 1) / (m - 1) - n;

**return** p;

}

O(n)

J(m, n) = [(J(m, n – 1) + m – 1) mod n] + 1

## Eratosthenes Sieve

**template**<**int** **MAXP**> **struct** BitSieve {

**#define** isOn(x) ( p[x >> 6] & ( 1 << ( (x & 63) >> 1) ) )

**#define** turnOn(x) ( p[x >> 6] |= ( 1 << ( (x & 63) >> 1 ) ) )

**int** p[(**MAXP** >> 6) + 1];

**BitSieve**() {

**for** (**int** i = 3; i \* i <= **MAXP**; i += 2) {

**if** (!isOn(i)) {

**int** ii = i \* i, i2 = i << 1;

**for** (**int** j = ii; j <= **MAXP**; j += i2) turnOn(j);

}

}

}

**bool** **operator []** (**const** **int** x) {

**return** x > 1 && (x == 2 || ( (x & 1) && !isOn(x) ) );

}

};

## Misof‘s Tree

**const** **int** MAXN = 20;

**int** tr[MAXN + 1][1 << MAXN];

**void** **insert**(**int** x) { For(i, 0, MAXN) tr[i][x]++, x >>= 1; }

**void** **erase**(**int** x) { For(i, 0, MAXN) tr[i][x]--, x >>= 1; }

**int** **kThElement**(**int** k) {

**int** a = 0, b = MAXN;

**while** (b--) a <<= 1, k -= (tr[b][a] < k ? tr[b][a++] : 0);

**return** a;

}

## Treap

**struct** treap {

**int** key, prio, size;

treap \*l, \*r;

**treap**() {

}

**treap**(**int** \_key, **int** \_prio, treap\* \_l, treap\* \_r) {

key = \_key; prio = \_prio; size = 1;

l = \_l; r = \_r;

**if** (l) size += l->size;

**if** (r) size += r->size;

/\*TODO\*/

}

};

**int** seed;

**inline** **int** **randPrio**() {

seed = (seed \* 1001 + 100621) % 999983;

**return** seed;

}

**inline** **int** **size**(treap\* &t) {

**return** t ? t->size : 0;

}

**inline** **int** **prio**(treap\* &t) {

**return** t ? t->prio : -1;

}

**inline** **void** **upd**(treap\* &t) {

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

/\*TODO\*/

}

treap\* **search**(treap\* &t, **int** pos) {

**if** (pos == size(t->l) + 1) **return** t;

**if** (pos <= size(t->l)) **return** search(t->l, pos);

**return** search(t->r, pos - size(t->l) - 1);

}

**inline** **void** **rotl**(treap\* &t) {

treap\* l = t->l; t->l = l->r; l->r = t; t = l;

upd(t->r); upd(t);

}

**inline** **void** **rotr**(treap\* &t) {

treap\* r = t->r; t->r = r->l; r->l = t; t = r;

upd(t->l); upd(t);

}

**inline** **void** **balance**(treap\* &t) {

**if** (prio(t->l) > t->prio) rotl(t);

**else** **if** (prio(t->r) > t->prio) rotr(t);

}

**void** **insert**(treap\* &t, **int** key, **int** prio, **int** pos) {

**if** (!t) {

t = **new** treap(key, prio, 0, 0);

**return**;

}

t->size++; /\*TODO\*/

**if** (pos > size(t)) insert(t->r, key, prio, pos);

**else** **if** (pos <= size(t->l) + 1) insert(t->l, key, prio, pos);

**else** insert(t->r, key, prio, pos - size(t->l) - 1);

balance(t);

}

treap\* **merge**(treap\* l, treap\* r) {

**if** (!l) **return** r;

**if** (!r) **return** l;

treap\* res;

**if** (l->prio < r->prio) {

treap\* t = merge(l, r->l);

res = **new** treap(r->key, r->prio, t, r->r);

**delete** r;

} **else** {

treap\* t = merge(l->r, r);

res = **new** treap(l->key, l->prio, l->l, t);

**delete** l;

}

**return** res;

}

**void** **erase**(treap\* &t, **int** pos) {

**if** (pos == size(t->l) + 1) {

**if** (!t->l && !t->r) {

**delete** t;

t = 0;

} **else** t = merge(t->l, t->r);

**return**;

}

t->size--; /\*TODO\*/

**if** (pos <= size(t->l)) erase(t->l, pos);

**else** erase(t->r, pos - size(t->l) - 1);

}

**inline** **void** **split**(treap\* t, treap\* &l, treap\* &r, **int** pos) {

insert(t, -1, inf, pos);

l = t->l; r = t->r;

**delete** t;

}

**inline** **void** **split**(treap\* t, treap\* &t1, treap\* &t2, treap\* &t3, **int** pos1, **int** pos2) {

treap\* temp;

split(t, t1, t2, pos1);

split(t2, t2, t3, pos2 - pos1 + 2);

}

**void** **dump**(treap\* node) {

**if** (node) {

dump(node->l);

/\* **TODO** \*/

dump(node->r);

}

}

## Left-Leaning Red-Black Tree

**struct** rbtree {

**struct** node {

**int** key, size, red, alive;

node \*l, \*r;

**void** **update**() { size = alive + l->size + r->size; }

} \*root, \*nil;

node \***create**(**int** key) {

node \*x = **new** node();

x->key = key; x->size = 1;

x->red = x->alive = 1;

x->l = x->r = nil;

**return** x;

}

**rbtree**() {

root = nil = **new** node();

nil->size = 0; nil->red = nil->alive = 0;

nil->l = nil->r = nil;

}

node \***rotate**(node \*x, node \*y) {

**if** (y == x->l) { x->l = y->r; y->r = x; }

**else** { x->r = y->l; y->l = x; };

y->red = x->red; x->red = 1;

x->update(); y->update();

**return** y;

}

node \***insert**(node \*x, **int** &key) {

**if** (x == nil) **return** create(key);

**if** (x->l->red && x->r->red) {

x->l->red = x->r->red = 0;

x->red ^= 1;

}

**if** (key == x->key) x->alive = 1;

**else** key < x->key ? x->l = insert(x->l, key) : x->r = insert(x->r, key);

x->update();

**if** (x->r->red) x = rotate(x, x->r);

**if** (x->l->red && x->l->l->red) x = rotate(x, x->l);

**return** x;

}

**void** **erase**(node \*x, **int** &key) {

**if** (x == nil) **return**;

**if** (key == x->key) x->alive = 0;

**else** key < x->key ? erase(x->l, key) : erase(x->r, key);

x->update();

}

**int** **size**() { **return** root->size; }

**void** **insert**(**int** key) { root = insert(root, key); }

**void** **erase**(**int** key) { erase(root, key); }

**int** **count\_lower**(**int** key) {

node \*x = root;

**int** total = 0;

**while** (x != nil) {

**if** (x->key >= key) x = x->l;

**else** {

total += x->l->size + x->alive;

x = x->r;

}

}

**return** total;

}

**int** **find\_kth**(**int** k) {

node \*x = root;

**int** pos = 0;

**while** (x != nil) {

pos = x->l->size + x->alive;

**if** (pos == k && x->alive) **break**;

**if** (pos >= k) x = x->l;

**else** {

k -= pos;

x = x->r;

}

}

**return** x->key;

}

};

## Splay Tree

**struct** Node {

**int** key, size;

**bool** reversed;

Node \*parent, \*left, \*right;

} \*root, \*nil;

Node \*node[maxn + 1];

**int** n, a[maxn + 1];

**void** **link**(Node \*parent, Node \*child, **bool** left) {

child->parent = parent;

left ? parent->left = child : parent->right = child;

}

**void** **update**(Node \*node) {

node->size = 1 + node->left->size + node->right->size;

}

**void** **build**() {

nil = node[0] = **new** Node(); nil->size = 0;

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

node[i] = **new** Node();

link(node[i], node[i - 1], **true**);

node[i]->key = a[i];

node[i]->right = nil;

node[i]->size = i;

node[i]->reversed = **false**;

}

node[n]->parent = nil;

root = node[n];

}

**void** **push**(Node \*node) {

**if** (node == nil || !node->reversed) **return**;

node->reversed = **false**;

swap(node->left, node->right);

node->left->reversed ^= **true**;

node->right->reversed ^= **true**;

}

Node \***find**(Node \*node, **int** i) {

**int** order;

**while** (**true**) {

push(node);

order = node->left->size + 1;

**if** (i == order) **return** node;

**if** (i < order) node = node->left;

**else** { node = node->right; i -= order; }

}

**return** 0;

}

**void** **uptree**(Node \*x) {

Node \*y = x->parent;

Node \*z = y->parent, \*beta;

**if** (x == y->left) {

beta = x->right;

link(y, beta, **true**);

link(x, y, **false**);

}

**else** {

beta = x->left;

link(y, beta, **false**);

link(x, y, **true**);

}

link(z, x, z->left == y);

update(y);

update(x);

}

**void** **splay**(Node \*x) {

Node \*y, \*z;

**while** (**true**) {

y = x->parent;

**if** (y == nil) **break**;

z = y->parent;

**if** (z != nil) {

**if** ((z->left == y) == (y->left == x)) uptree(y);

**else** uptree(x);

}

uptree(x);

}

}

**void** **split**(Node \*tree, **int** i, Node \*&left, Node \*&right) {

**if** (i == 0) {

left = nil;

right = tree;

**return**;

}

Node \*x = find(tree, i);

splay(x);

right = x->right; right->parent = nil;

left = x; left->right = nil; update(left);

}

Node \***join**(Node \*left, Node \*right) {

**if** (left == nil) **return** right;

**if** (right == nil) **return** left;

push(right);

**while** (**true**) {

push(left);

**if** (left->right == nil) **break**;

left = left->right;

}

splay(left);

link(left, right, **false**);

update(left);

**return** left;

}

**void** **visit**(Node \*node) {

**if** (node == nil) **return**;

push(node);

visit(node->left);

**printf**("%d ", node->key);

visit(node->right);

}

**void** **rotate**(**int** l, **int** r) {

Node \*x, \*y, \*z, \*t;

split(root, r, t, z);

split(t, l - 1, x, y);

y->reversed ^= **true**;

t = join(x, y);

root = join(t, z);

}

## Heavy-light decomposition

/\* vertex: 1-based | edge: lis[] \*/

**struct** SegmentTree {

SegmentTree \*dad;

**int** p, size;

vector<**int**> h, arr; /\* segment array \*/

**SegmentTree**() { size = 0; dad = NULL; p = 0; }

**SegmentTree**(SegmentTree \*dad, **int** h) {

size = 0; **this**->dad = dad; p = h;

}

**void** **update**(**int** l, **int** r, **int** u, **int** v, **int** node) {}

};

list<**int**> lis[maxn];

list<SegmentTree> lisTree;

**int** h[maxn], f[maxn][13], posIn[maxn], posOut[maxn], cntIn, cntOut, num[maxn];

SegmentTree \*ori[maxn];

**inline** **int** **getDeg**(**int** l) {

**return** **floor**(log(l) / log(2) + 1e-9);

}

**inline** **bool** **isDad**(**int** u, **int** v) {

**return** posIn[u] <= posIn[v] && posOut[u] >= posOut[v];

}

**int** **getLca**(**int** u, **int** v) {

**if** (isDad(u, v)) **return** u;

**if** (isDad(v, u)) **return** v;

Ford(i,12,0) **if** (f[u][i] != -1 && !isDad(f[u][i], v)) u = f[u][i];

**return** f[u][0];

}

**void** **initialize**(**int** u) {

posIn[u] = ++cntIn;

**if** (u != 1) {

**int** l = getDeg(h[u] - 1);

For(i,1,l) f[u][i] = f[f[u][i - 1]][i - 1];

}

num[u] = 1;

**int** v;

Fit(it,lis[u]) {

v = \*it;

**if** (h[v] != 0) **continue**;

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

f[v][0] = u;

initialize(v);

num[u] += num[v];

}

posOut[u] = ++cntOut;

}

**void** **addTree**(**int** u, SegmentTree &tree) {

tree.h.pb(h[u]);

tree.size++;

ori[u] = &tree;

**int** v, pos = -1, minNum = -1;

Fit(it,lis[u]) {

v = \*it;

**if** (h[v] < h[u]) **continue**;

**if** (num[v] > minNum) {

minNum = num[v];

pos = v;

}

}

**if** (pos == -1) **return**;

addTree(pos, tree);

Fit(it,lis[u]) {

v = \*it;

**if** (h[v] < h[u] || v == pos) **continue**;

lisTree.pb(SegmentTree(&tree, h[u]));

addTree(v, lisTree.back());

}

}

**void** **build**() {

lisTree.clear();

ms(h, 0); ms(f, -1);

cntIn = 0; cntOut = 0;

ms(posIn, 0); ms(posOut, 0);

h[1] = 1;

initialize(1);

lisTree.pb(SegmentTree());

addTree(1, lisTree.back());

Fit(it,lisTree) {

/\* Resize segment array \*/

it->arr.resize(it->size \* 4, inf);

}

}

**void** **process**(**int** u, **int** v) {

SegmentTree \*tree;

**int** lca = getLca(u, v);

**if** (u != lca) {

tree = ori[u];

**int** l = h[lca] + 1, r = h[u];

**do** {

/\* process \*/

tree->update(max(l, tree->h[0]) - tree->h[0], r - tree->h[0], 0, tree->size - 1, 1);

/\* end \*/

**if** (l >= tree->h[0]) **break**;

r = tree->p;

tree = tree->dad;

} **while** (**true**);

}

**if** (v != lca) {

tree = ori[v];

**int** l = h[lca] + 1, r = h[v];

**do** {

/\* process \*/

tree->update(max(l, tree->h[0]) - tree->h[0], r - tree->h[0], 0, tree->size - 1, 1);

/\* end \*/

**if** (l >= tree->h[0]) **break**;

r = tree->p;

tree = tree->dad;

} **while** (**true**);

}

}

## Hình học 2D

**struct** Point {

ld x, y;

**bool** **operator ==**(**const** Point& that) **const** {

**return** (cmp(x, that.x) == 0 && cmp(y, that.y) == 0);

}

**bool** **operator <**(**const** Point& that) **const** {

**if** (cmp(x, that.x) != 0) **return** cmp(x, that.x) < 0;

**return** cmp(y, that.y) < 0;

}

};

## So sánh số thực, sử dụng epsilon

**int** **cmp**(ld a, ld b) {

**if** (abs(a - b) < eps) **return** 0;

**return** a > b ? 1 : -1;

}

## Hàm CCW

Đi từ p0 -> p1 -> p2: +1 rẽ trái, -1 rẽ phải, 0 thẳng hàng

**int** **ccw**(Point p0, Point p1, Point p2) {

ld dx1 = p1.x - p0.x, dy1 = p1.y - p0.y;

ld dx2 = p2.x - p0.x, dy2 = p2.y - p0.y;

**return** cmp(dx1 \* dy2 - dx2 \* dy1, 0);

}

## Tìm tiếp tuyến qua 2 đường tròn (2 đường tròn không trùng nhau)

**struct** Circle : Point { ld r; };

**struct** Line { ld a, b, c; };

**void** **tangents**(Point c, ld r1, ld r2, vector<Line> &ans) {

ld r = r2 - r1;

ld z = sqr(c.x) + sqr(c.y);

ld d = z - sqr(r);

**if** (d < -eps) **return**;

d = sqrt(abs(d));

Line l;

l.a = (c.x \* r + c.y \* d) / z;

l.b = (c.y \* r - c.x \* d) / z;

l.c = r1;

ans.pb(l);

}

vector<Line> **tangents**(Circle a, Circle b) {

vector<Line> ans;

**for** (**int** i = -1; i <= 1; i += 2)

**for** (**int** j = -1; j <= 1; j += 2)

tangents(b - a, a.r \* i, b.r \* j, ans);

Rep(i,sz(ans)) ans[i].c -= ans[i].a \* a.x + ans[i].b \* a.y;

**return** ans;

}

## Tính phương trình đường thẳng đi qua 2 điểm: ax + by + c = 0

Hai điểm p0 và p1 không được trùng nhau

**void** **getLine**(Point p0, Point p1, ld& a, ld& b, ld& c) {

a = p1.y - p0.y;

b = p0.x - p1.x;

c = -(a \* p0.x + b \* p0.y);

}

## Tìm giao điểm của 2 đường thẳng có phương trình cho trước

Trả ra -1 nếu trùng nhau, 0 nếu song song, +1 nếu cắt nhau tại điểm P

**int** **lineIntersect**(ld a0, ld b0, ld c0, ld a1, ld b1, ld c1, Point& P) {

ld d = a0 \* b1 - a1 \* b0;

ld dx = b0 \* c1 - b1 \* c0;

ld dy = -(c1 \* a0 - c0 \* a1);

**if** (cmp(d, 0) == 0) **return** (cmp(dx, 0) == 0 && cmp(dy, 0) == 0) ? -1 : 0;

P.x = dx / d;

P.y = dy / d;

**return** 1;

}

## Kiểm tra điểm có nằm hoàn toàn trong đa giác lồi hay không

Các điểm a[0..n-1] theo chiều kim đồng hồ, điểm nằm trên cạnh không tính là nằm trong.

**bool** **strictlyInside**(Point O, Point a[], **int** n) {

**int** low = 1, high = n - 1, mid, ret = 1;

**while** (low <= high) {

mid = (low + high) >> 1;

**if** (ccw(a[0], O, a[mid]) > 0) low = mid + 1;

**else** {

ret = mid;

high = mid - 1;

}

}

**return** ret > 1 && ccw(a[0], O, a[n - 1]) < 0 && ccw(a[ret], O, a[ret - 1]) < 0;

}

## Tìm bao lồi của một tập điểm 2D

Kết quả trả về theo chiều ngược chiều kim đồng hồ

Point O;

**bool** **cmp\_angle**(Point P1, Point P2) {

**int** ret = ccw(O, P1, P2);

**if** (ret != 0) **return** ret > 0;

**return** sqrdist(O, P1) < sqrdist(O, P2);

}

Các điểm đánh số từ 0 -> n-1

**int** **convex\_hull**(Point a[], **int** n) {

**if** (n <= 3) **return** n;

**int** imin = 0;

Rep(i, n) **if** (a[i].y < a[imin].y || (a[i].y == a[imin].y && a[i].x < a[imin].x)) imin = i;

swap(a[imin], a[0]);

O = a[0];

sort(a + 1, a + n, cmp\_angle);

**int** m = 2;

For(i, 2, n - 1) {

**while** (m > 1 && ccw(a[i], a[m - 1], a[m - 2]) >= 0) --m;

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

++m;

}

**return** m;

}

## Tính cos của góc giữa 2 vector [p0,p1] và [p0,p2]

**ld** **cosAngle**(Point p0, Point p1, Point p2) {

**ld** d1 = dist(p0, p1);

**ld** d2 = dist(p0, p2);

**return** ((p1.x - p0.x) \* (p2.x - p0.x) + (p1.y - p0.y) \* (p2.y - p0.y)) / (d1 \* d2);

}

## Tìm giao của 2 đường tròn

**Kết quả lưu trong vector vp, trả về số giao điểm (-1 nếu 2 đường tròn trùng nhau)**

**int** two**CircleIntersect**(Point p0, ld r0, Point p1, ld r1, vector<Point>& vp) {

vp.clear();

ld d = dist(p0, p1);

**if** (d > r0 + r1) **return** 0;

**if** (d < abs(r0 - r1)) **return** 0;

**if** (cmp(r0 - r1, 0) == 0 && cmp(p0.x - p1.x, 0) == 0 && cmp(p0.y - p1.y, 0) == 0) **return** -1;

ld a = (r0 \* r0 - r1 \* r1 + d \* d) / 2 / d;

ld h = sqrt(r0 \* r0 - a \* a);

Point p2(p0.x + a \* (p1.x - p0.x) / d, p0.y + a \* (p1.y - p0.y) / d);

vp.pb(Point(p2.x + h \* (p1.y - p0.y) / d, p2.y - h \* (p1.x - p0.x) / d));

**if** (cmp(h, 0)) **return** 1;

vp.pb(Point(p2.x - h \* (p1.y - p0.y) / d, p2.y + h \* (p1.x - p0.x) / d));

**return** 2;

}

## Tìm giao điểm của đường thẳng và đường tròn

**Kết quả lưu trong vector p, trả về số giao điểm**

**int** **lineCircleIntersect**(Point p0, **ld** r0, Point p1, Point p2, vector<Point>& p) {

p.clear();

Point u = p2 - p1;

Point proj = projection(p0 - p1, u) + p1;

**if** ((proj - p0).length() < r0 - eps) {

ld add = sqrt(r0 \* r0 - (proj - p0) \* (proj - p0));

u = u \* (add / u.length());

p.pb(proj + u); p.pb(proj - u);

**return** 2;

} **else** **if** (abs((proj - p0).length() - r0) <= eps) {

p.pb(proj); **return** 1;

} **else**

**return** 0;

}

## Tính trung trực của đoạn thằng [p0, p1]

**bool** **perpendicularBisector**(Point p0, Point p1, ld& a, ld& b, ld& c) {

**if** (p0 == p1) **return** **false**;

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

Point p2 = Point((p0.x + p1.x) / 2, (p0.y + p1.y) / 2);

c = -(p2.x \* a + p2.y \* b);

**return** **true**;

}

## Cắt đa giác

**Cắt đa giác a[0..n-1] bằng đường thẳng đi qua 2 điểm PQ, giữ lại phần đa giác ở cùng phía với điểm O.**

**void** **cutPolygon**(Point a[], **int** &n, Point P, Point Q, Point O) {

vector<Point> b;

**int** co = ccw(O, P, Q);

ld a1, b1, c1;

ld a2, b2, c2;

lineEquation(P, Q, a1, b1, c1);

Point I;

Rep(i, n) {

**int** j = (i + 1) % n;

lineEquation(a[i], a[j], a2, b2, c2);

**if** (ccw(a[i], P, Q) \* co < 0) {

**if** (ccw(a[j], P, Q) \* co > 0) {

lineIntersect(a1, b1, c1, a2, b2, c2, I);

b.pb(I);

}

} **else** {

b.pb(a[i]);

**if** (ccw(a[i], P, Q) \* co > 0 && ccw(a[j], P, Q) \* co < 0) {

lineIntersect(a1, b1, c1, a2, b2, c2, I);

b.pb(I);

}

}

}

n = sz(b);

Rep(i, n) a[i] = b[i];

}

## Tìm đường tròn đi qua 3 điểm

Trả về FALSE nếu 3 điểm thẳng hàng, ngược lại trả về TRUE

**bool** **getCircle**(Point p0, Point p1, Point p2, Point& p3, ld& r) {

**if** (ccw(p0, p1, p2) == 0) **return** **false**;

ld a0, b0, c0, a1, b1, c1;

getCenterLine(p0, p1, a0, b0, c0);

getCenterLine(p0, p2, a1, b1, c1);

ld d = a0 \* b1 - a1 \* b0;

ld dx = b0 \* c1 - b1 \* c0;

ld dy = -(c1 \* a0 - c0 \* a1);

p3.x = dx / d; p3.y = dy / d;

r = dist(p3, p0);

**return** **true**;

}

## Chia đa giác bất kỳ thành các tam giác

Triangle **calTriangle**(vector<Point> &P){

**if**(sz(P) == 3) **return** Triangle(P[0], P[1], P[2]);

**int** num = sz(P), u, v, t;

ld S = area(P);

Rep(i, num){

vector<Point> temp;

u = (i + 1) % num; v = (i + 2) % num;

Rep(j, num) **if**(j != u) temp.pb(P[j]);

**bool** flag = **true**;

Rep(j, num){

t = (j + 1) % num;

**if**(i != j && i != t && v != j && v != t && isSegmentCut(P[i], P[v], P[j], P[t])){

flag = **false**;

**break**;

}

}

vector<Point> temp1; temp1.pb(P[i]); temp1.pb(P[u]); temp1.pb(P[v]);

**if**(flag && cmp(area(temp) + area(temp1), S) == 0){

Triangle T = Triangle(P[i], P[u], P[v]);

P = temp;

**return** T;

}

}

}

## TRIPOD

Tìm đường tròn nhỏ nhất chứa 3 điểm trong n điểm cho trước

**void** **run**(**int** l, **int** r) {

**if** (r - l <= 1) **return**;

**if** (r - l == 2) {

res = min(res, radius(p[l], p[l+1], p[r]));

**return**;

}

**int** m = (l + r) / 2;

run(l, m);

run(m + 1, r);

vector <point> a;

**for** (**int** i = m; i >= l && p[m].X - 2\*res <= p[i].X; -- i) a.push\_back(p[i]);

**for** (**int** i = m+1; i <= r && p[m].X + 2\*res >= p[i].X; ++ i) a.push\_back(p[i]);

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

**int** na = a.size();

**for** (**int** i = 0; i < na; ++ i)

**for** (**int** j = i + 1; j < na && a[i].Y + 2\*res >= a[j].Y; ++ j)

**for** (**int** k = j + 1; k < na && a[i].Y + 2\*res >= a[k].Y; ++ k)

res = min(res, radius(a[i], a[j], a[k]));

}

## Tìm 2 điểm gần nhau nhất trong tập điểm

**struct** YComparator {

**bool** **operator ()**(**const** Point p0, **const** Point p1) **const** {

**if** (p0.y != p1.y) **return** p0.y < p1.y;

**return** p0.x < p1.x;

}

};

ld **closestPairDist**(Point a[], **int** n) {

sort(a, a + n);

set<Point, YComparator> b;

**int** j = 0;

ld ret = 1e100;

Rep(i, n) {

**while** (a[i].x - a[j].x > ret) b.erase(a[j++]);

Point c = a[i];

c.y -= ret;

set<Point, YComparator>::iterator lowerIt = b.lower\_bound(c);

c.y += 2 \* ret;

set<Point, YComparator>::iterator upperIt = b.upper\_bound(c);

**for** (set<Point, YComparator>::iterator it = lowerIt; it != upperIt; it++) {

ret = min(ret, dist(a[i], \*it));

}

b.insert(a[i]);

}

**return** ret;

}

## Hình học 3D

## Ma trận phép xoay quanh trục

Quay điểm P quanh trục vector đơn vị (x,y,z) 1 góc a

[ txx + c | txy – sz | txz + sy ]

[ txy + sz | tyy + c | tyz – sx ]

[ txz - sy | tyz + sx | tzz + c ]

Với: c = cos(a) s = sin(a) t = 1 – cos(a)

**inline** **ld** **det**(**ld** a, **ld** b, **ld** c, **ld** d) {

**return** a \* d - b \* c;

}

**struct** Point {

ld x, y, z;

ld **length**() {

**return** sqrt(x \* x + y \* y + z \* z);

}

Point **operator %**(**const** Point &op) **const** {

**return** Point(det(y, z, op.y, op.z), -det(x, z, op.x, op.z), det(x, y,

op.x, op.y));

}

};

**struct** Space {

ld a, b, c, d;

**Space**(Point p0, Point p1, Point p2) {

a = p0.y \* (p1.z - p2.z) + p1.y \* (p2.z - p0.z) + p2.y \* (p0.z - p1.z);

b = p0.z \* (p1.x - p2.x) + p1.z \* (p2.x - p0.x) + p2.z \* (p0.x - p1.x);

c = p0.x \* (p1.y - p2.y) + p1.x \* (p2.y - p0.y) + p2.x \* (p0.y - p1.y);

d = -p0.x \* (p1.y \* p2.z - p2.y \* p1.z) - p1.x \* (p2.y \* p0.z - p0.y \* p2.z)

- p2.x \* (p0.y \* p1.z - p1.y \* p0.z);

}

};

Point **projection**(Point v, Point u) { // Chiếu vector v lên vector u

ld scalar = (v \* u) / (u \* u);

**return** u \* scalar;

}

Point **projection**(Point p, Point a, Point b, Point c) { // Chiếu điểm p lên mặt phẳng ABC

Point u = (b - a) % (c - a), v = p - a;

ld scalar = (v \* u) / (u \* u);

**return** p - (u \* scalar);

}

ld **dist**(Point p, Point a, Point b) { // Khoảng cách từ p tới đường thẳng AB

p = p - a;

Point proj = projection(p, b - a);

**return** sqrt(p \* p - proj \* proj);

}

ld **area**(Point a, Point b, Point c) { // Diện tích tam giác ABC

ld h = dist(a, b, c);

**return** (h \* (b - c).length()) / 2;

}

ld **volume**(Point x, Point y, Point z) { // Thể tích của 3 vector

Point base = Point(y.y \* z.z - y.z \* z.y, y.z \* z.x - y.x \* z.z, y.x \* z.y

- y.y \* z.x);

**return** abs(x.x \* base.x + x.y \* base.y + x.z \* base.z) / 3;

}

## 3D Convex hull

V - E + F = 2 | E <= 3V – 6 | F <= 2V - 4

**inline** **int** **sign**(ld x) { **return** x < -eps ? -1 : x > eps ? 1 : 0; }

vector<Point> arr;

vector<**int**> rnd;

set<**int**> used;

Side **getFirstSide**(vector<Point> &p) {

**int** i1 = 0;

Rep(i,sz(p)) {

**if** (p[i].z < p[i1].z || (p[i].z == p[i1].z && p[i].x < p[i1].x)

|| (p[i].z == p[i1].z && p[i].x == p[i1].x && p[i].y < p[i1].y)) {

i1 = i;

}

}

**int** i2 = i1 == 0 ? 1 : 0;

Rep(i,sz(p)) {

**if** (i != i1) {

Point zDir(0, 0, 1);

ld curCos = (p[i] - p[i1]) \* zDir / (p[i] - p[i1]).length();

ld bestCos = (p[i2] - p[i1]) \* zDir / (p[i2] - p[i1]).length();

**if** (curCos < bestCos) {

i2 = i;

}

}

}

**int** i3 = -1;

**int** n = sz(p);

Rep(ri,n) {

**int** i = rnd[ri];

**if** (i != i1 && i != i2) {

Point norm = (p[i1] - p[i]) % (p[i2] - p[i]);

**bool** sg[] = { 0, 0, 0 };

Rep(t,n) {

**int** j = rnd[t];

sg[1 + sign((p[j] - p[i]) \* norm)] = **true**;

**if** (sg[0] && sg[2]) {

**break**;

}

}

**if** (sg[0] ^ sg[2]) {

i3 = i;

**if** (!sg[0]) {

swap(i3, i2);

}

**break**;

}

}

}

vector<**int**> res;

res.pb(i1);

res.pb(i2);

res.pb(i3);

**return** res;

}

**inline** **int** **getSideKey**(**int** i, **int** j, **int** k) {

**int** key = (i \* 1000 + j) \* 1000 + k;

**return** key;

}

**inline** **bool** **isUsed**(**int** i, **int** j, **int** k) {

**return** used.find(getSideKey(i, j, k)) != used.end();

}

**inline** ld **getAngle**(**const** Point &n1, **const** Point &n2) {

**return** atan2((n1 % n2).length(), n1 \* n2);

}

**inline** ld **getNormsAngle**(**int** i, **int** j, **int** k, **int** t, vector<Point> &p) {

Point n1 = (p[j] - p[i]) % (p[k] - p[i]);

Point n2 = (p[t] - p[i]) % (p[j] - p[i]);

**return** getAngle(n1, n2);

}

**void** **dfs**(**int** i, **int** j, **int** k, vector<Point> &p, vector<Side> &sides) {

**if** (i < j && i < k) {

vector<**int**> side(3);

side[0] = i;

side[1] = j;

side[2] = k;

sides.pb(side);

}

**int** key = getSideKey(i, j, k);

used.insert(key);

**int** n = sz(p);

**if** (!isUsed(j, k, i))

dfs(j, k, i, p, sides);

**if** (!isUsed(k, i, j))

dfs(k, i, j, p, sides);

**int** bestT = -1;

ld bestAngle = 1e20;

Point curNorm = (p[j] - p[i]) % (p[k] - p[i]);

Point dir = p[j] - p[i];

Rep(t,n) {

**if** (t != i && t != j && t != k) {

Point newNorm = (p[t] - p[i]) % dir;

ld curAng = curNorm \* newNorm / newNorm.length();

**if** (bestT == -1 || curAng > bestAngle) {

bestT = t;

bestAngle = curAng;

}

}

}

**if** (!isUsed(i, bestT, j)) {

dfs(i, bestT, j, p, sides);

}

}

vector<Side> **convexHull3d**(vector<Point> p) {

used.clear();

rnd.resize(sz(p));

Rep(i,sz(p))

rnd[i] = i;

random\_shuffle(rnd.begin(), rnd.end());

Side side0 = getFirstSide(p);

vector<Side> sides;

dfs(side0[0], side0[1], side0[2], p, sides);

**return** sides;

}

/\* eliminate conflict sides \*/

**inline** **bool** **isEmpty**(Point x, Point y, Point z) {

**return** abs(x \* Point(y.y \* z.z - y.z \* z.y, y.z \* z.x - y.x \* z.z, y.x

\* z.y - y.y \* z.x)) <= eps;

}

**inline** **bool** **conflict**(Side a, Side b) {

Point x = arr[a[0]], y = arr[a[1]], z = arr[a[2]];

Rep(i,3) {

Point t = arr[b[i]];

**if** (!isEmpty(x - t, y - t, z - t))

**return** **false**;

}

**return** **true**;

}

vector<Side> **eliminate**(vector<Side> p) {

vector<Side> res;

vector<**bool**> fre;

fre.resize(sz(p), **true**);

Rep(i,sz(p)) {

**if** (!fre[i])

**continue**;

res.pb(p[i]);

For(j,i+1,sz(p) - 1)

**if** (fre[j]) {

**if** (conflict(p[i], p[j])) {

fre[j] = **false**;

res.back().insert(res.back().end(), p[j].begin(),

p[j].end());

}

}

}

Rep(i,sz(res)) {

sort(res[i].begin(), res[i].end());

res[i].resize(unique(res[i].begin(), res[i].end()) - res[i].begin());

}

**return** res;

}

## Infix order to Postfix order (RPN)

|  |  |
| --- | --- |
| ( | 0 |
| +, - | 1 |
| \*, / | 2 |

Stack.Clear;

For (T = các phần tử đọc từ INFIX){

Switch (T) {

Case ‘(’: Stack.Push(T);

Case ‘)’: Do {

X = Stack.Pop;

If (X <> ’(‘) Print(X);

}While(X != ’(‘);

Case +, -, \*, /: {

While (!Stack.Empty && Priority(T) <= Priority(Stack.Top)) Print(Stack.Pop);

Stack.Push(T);

}

Default: Print(T);

}

}

While (!Stack.Empty) Print(Stack.Pop);

## Gauss Elimination

**int** **gauss**(vector<vector<ld> > a, vector<ld> &ans){

**int** n = sz(a);

**int** m = (sz(a[0])) - 1;

vector<**int**> where(m, -1);

**for**(**int** col = 0, row = 0; col < m && row < n; ++col){

**int** sel = row;

**for**(**int** i = row; i < n; i++) **if**(abs(a[i][col]) > abs(a[sel][col])) sel = i;

**if**(abs(a[sel][col]) < eps) **continue**;

For(i, col, m) swap(a[row][i], a[sel][i]);

where[col] = row;

Rep(i, n) **if**(i != row){

ld c = a[i][col] / a[row][col];

For(j, col, m) a[i][j] -= a[row][j] \* c;

}

++row;

}

ans.assign(m, 0);

Rep(i, m) **if**(where[i] != -1) ans[i] = a[where[i]][m] / a[where[i]][i];

Rep(i, n){

ld sum = 0;

Rep(j, m) sum += a[i][j] \* ans[j];

**if**(abs(sum - a[i][m]) > eps) **return** 0;

}

Rep(i, m) **if**(where[i] == -1) **return** inf;

**return** 1;

}

## Matrix Power Sum

s = a^1 + a^2 + ... + a^k

ll t[maxn][maxn], m[maxn][maxn];

**void** **xpowersum**(ll a[maxn][maxn], **int** k, ll s[maxn][maxn]) {

**if** (k == 0) {

Rep(i, n) Rep(j, n) s[i][j] = m[i][j] = 0;

**return**;

}

**if** (k == 1) {

Rep(i, n) Rep(j, n) s[i][j] = m[i][j] = a[i][j];

**return**;

}

xpowersum(a, k / 2, s);

xmul(s, m, t);

xadd(s, t, s);

xmul(m, m, t);

**if** (k & 1) {

xmul(t, a, m);

xadd(s, m, s);

} **else** {

Rep(i, n) Rep(j, n) m[i][j] = t[i][j];

}

}

## Brucker’s Algorithm

Thuật toán Brucker được sử dụng để tăng tốc thuật toán QHĐ đối với các bài toán có dạng:  
- Cần tính một hàm F(i), i = 1..N  
- F(i) = min{F(j) + c(i,j)}  
- Định lý:  
   o Đặt Fi(j) = F(j) + c(i,j)  
      => F(i) = min{Fi(j)}  
   o Nếu ta có thể tìm được 2 hàm g và f thỏa mãn:  
         f(i) ≥ f(i+1) với i = 1..N-1 (f là hàm giảm)  
        và Fi(j) ≤ Fi(k) ↔ g(j,k) ≤ f(i)

Thì có thể sử dụng thuật toán Brucker để giải quyết bài toán.

Bài toán 1 (GROUP): Cho n ≤ 300000 cặp số (x,y) (1 ≤ x, y ≤ 1000000). Ta có thể nhóm một vài cặp số lại thành một nhóm. Giả sử một nhóm gồm các cặp số thứ a1, a2, ..., am thì chi phí cho nhóm này sẽ là max(xa1, xa2, ..., xam) \* max(ya1, ya2, ..., yam). Tìm cách phân nhóm có tổng chi phí bé nhất.

Nếu tồn tại i và j thỏa mãn: xi ≤ xj và yi ≤ yj thì ta có thể loại bỏ cặp số (xi,yi) mà không làm thay đổi kết quả bài toán. Do đó, nếu ta sort tất cả các cặp số tăng theo xi, thì trong dãy đã sắp xếp, yi giảm dần.

Gọi F(i) = chi phí nhỏ nhất để phân nhóm các cặp số i..N  
=> F(i) = min{ F(j+1) + x(j) \* y(i) }

Đặt Fi(j) = F(j+1) + x(j) \* y(i), ta có:  
Fi(j) ≤ Fi(k)  
↔ F(j+1) + x(j) \* y(i) ≤ F(k+1) + x(k) \* y(i)  
↔ F(j+1) – F(k+1) ≤ y(i) \* ( x(k) – x(j) )

Ta chọn  
g(j,k) = ( F(j+1) – F(k+1) ) / ( x(k) – x(j) )  
f(i) = y(i)

// g(j, k) <= f(i) voi j < k

**bool** **isWorse1**(**int** j, **int** k, **int** i) {

**return** F[j + 1] - F[k + 1] <= (a[k].x - a[j].x) \* a[i].y;

}

// g(j, k) <= g(k, l) voi j < k < l

**bool** **isWorse2**(**int** j, **int** k, **int** l) {

**return** (F[j + 1] - F[k + 1]) \* (a[l].x - a[k].x) <= (F[k + 1] - F[l + 1]) \* (a[k].x - a[j].x);

}

ll **brucker**() {

front = back = 1; que[1] = n;

F[n + 1] = 0; F[n] = a[n].x \* a[n].y;

Ford(i, n - 1, 1) {

**while** (front + 1 <= back && isWorse2(i, que[back], que[back - 1])) --back;

que[++back] = i;

**while** (front + 1 <= back && isWorse1(que[front + 1], que[front], i)) ++front;

**int** j = que[front];

F[i] = F[j + 1] + a[j].x \* a[i].y;

}

**return** F[1];

}

ll **doit**() {

sort(p + 1, p + n + 1); // x tăng dần, cùng x thì y tăng dần

top = 0; s[++top] = 1;

For(i, 2, n) {

**while** (top > 0 && p[i].y >= p[s[top]].y) --top;

s[++top] = i;

}

n = top;

For(i, 1, n) a[i] = p[s[i]];

**return** brucker();

}

Bài toán 2 (NKLEAVES): Cho n đống lá tại các tọa độ x1, x2, .., xn. Mỗi đống lá có khối lượng là w1, w2, .., wn. Cần dồn n đống lá này thành k đống lá (k <= n) sao cho tổng chi phí là nhỏ nhất. Chỉ được phép dồn đống lá i đến vị trí của đống lá j với j > i, tốn chi phí là wi \* (xj – xi).

ll **func**(**int** level, **int** k, **int** l) {

**return** F[level][k + 1] - F[level][l + 1] + x[l] \* s[l + 1] - x[k] \* s[k + 1] - p[l + 1] + p[k + 1];

}

// g(k, l) <= f(i)

**bool** **isWorse1**(**int** level, **int** k, **int** l, **int** i) {

**return** func(level, k, l) <= (x[l] - x[k]) \* s[i];

}

// g(j, k) <= g(k, l)

**bool** **isWorse2**(**int** level, **int** j, **int** k, **int** l) {

**return** func(level, j, k) \* (x[l] - x[k]) <= func(level, k, l) \* (x[k] - x[j]);

}

ll **cost**(**int** i, **int** j) {

**return** (ll)x[j] \* (s[i] - s[j + 1]) + p[j + 1] - p[i];

}

ll **doit**() {

s[n + 1] = p[n + 1] = 0;

Ford(i, n, 1) {

s[i] = s[i + 1] + w[i];

p[i] = p[i + 1] + (ll)x[i] \* w[i];

}

F[1][n + 1] = 0;

For(i, 1, n) F[1][i] = cost(i, n);

**int** level = 1;

For(step, 2, K) {

level ^= 1;

For(i, 1, n + 1) F[level][i] = F[level ^ 1][i];

front = back = 1; que[1] = n;

Ford(i, n - 1, 1) {

**while** (front + 1 <= back && isWorse2(level ^ 1, i, que[back], que[back - 1])) --back;

que[++back] = i;

**while** (front + 1 <= back && isWorse1(level ^ 1, que[front + 1], que[front], i)) ++front;

**int** j = que[front];

upmin(F[level][i], F[level ^ 1][j + 1] + cost(i, j));

}

}

**return** F[level][1];

}

## NEWJ

Tìm tất cả số các nguyên x thoả mãn (x\*x) mod n = a mod n. Trong đó n là số nguyên tố và ước chung lớn nhất của a và n = 1, 0 ≤ x ≤ n – 1. (1 ≤ a, n ≤ 1000000)

ll a, n, mu2[50];

ll **mu**(ll u, ll k) {

**if** (k == 1) **return** u;

**else** {

ll t = mu(u, k / 2);

**if** (k % 2 == 0) **return** (t \* t) % n;

**else** **return** (t \* t \* u) % n;

}

}

ll **xuly**(ll a, ll p) {

**if** (a % n == 0) **return** 0;

**if** (mu(a, (p - 1) / 2) != 1) **return** -1;

ll m = p - 1, k = 0, z, c, kk, the;

**while** (m % 2 == 0) { k++; m /= 2; }

ll x = mu(a, (m + 1) / 2), b = mu(a, m);

**if** (b == 1) **return** x;

the = b;

**for** (z = 2;; z++) **if** (mu(z, (p - 1) / 2) == p - 1) **break**;

c = mu(z, m);

**while** (b != 1) {

the = b, kk = 0;

**while** (the != 1) {

kk++;

the = (the \* the) % p;

}

x = (mu(c, mu2[k - kk - 1]) \* x) % p;

b = (mu(c, mu2[k - kk]) \* b) % p;

}

**return** x;

}

**int** **main**() {

mu2[0] = 1;

**for** (**int** i = 1; i <= 40; i++) mu2[i] = mu2[i - 1] \* 2;

**scanf**("%lld %lld", &a, &n);

a = a % n;

**if** (a % n == 0) **printf**("0\n");

**else** **if** (n == 2) **printf**("1\n");

**else** {

ll x = xuly(a, n);

ll y = n - x;

**if** (x == -1) **printf**("Khong co\n");

**else** {

**if** (x > y) swap(x, y);

**printf**("%lld %lld\n", x, y);

}

}

**return** 0;

}

## SKIVER1 / PK11I

Cho đồ thị có hướng G, tìm tập gồm ít đường đi đơn nhất (có thể chung đỉnh/cạnh) để đảm bảo mỗi cạnh của đồ thị nằm trong ít nhất một đường đi trong tập.

**int** **main**() {

gi(n);

ms(indeg, 0);

ms(outdeg, 0);

**int** s = 0, t = n + 1;

dinic.init(n + 2, s, t);

res = 0;

**int** u, v;

Rep(i, n - 1) {

gi(v); gi(u);

++u; ++v;

++outdeg[u];

++indeg[v];

dinic.add(u, v, inf, 0);

}

For(u, 1, n) {

**if** (indeg[u] > outdeg[u]) dinic.add(s, u, indeg[u] - outdeg[u], 0);

**else** **if** (outdeg[u] > indeg[u]) dinic.add(u, t, outdeg[u] - indeg[u], 0);

}

For(u, 1, n) **if** (indeg[u] > outdeg[u]) res += indeg[u] - outdeg[u];

res -= dinic.maxflow();

**printf**("%lld\n", res);

**return** 0;

}

## ASSIGN4

There are m types of worker (1 .. m) and n types of task (1 .. n). There are a(i) workers of type #i and b(j) postitions for task #j. C(i, j) is the cost of hiring a worker of type #i to do the task of type #j. Your job is to minimize the cost of hiring workers to fill all the positions given that the total number of workers is equal to the total number of positions.

**int** qf, ql, q[MAX \* 2], p[MAX \* 2], x, y, m, n, gx[MAX], gy[MAX], a[MAX][MAX], fx[MAX], fy[MAX], cx[MAX], cy[MAX], f[MAX][MAX], ret;

**bool** **find**() {

qf = ql = 0;

q[0] = x;

**for** (**int** i = 0; i < m + n; i++) p[i] = -1;

p[x] = m + n;

**while** (qf <= ql) {

**int** i = q[qf++];

**if** (i < m) {

**for** (**int** j = m; j < m + n; j++)

**if** (p[j] < 0 && gx[i] + gy[j - m] == a[i][j - m]) {

q[++ql] = j;

p[j] = i;

}

} **else** {

**if** (fy[i - m] < cy[i - m]) {

y = i;

**return** **true**;

}

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

**if** (p[j] < 0 && f[j][i - m] > 0) {

q[++ql] = j;

p[j] = i;

}

}

}

**return** **false**;

}

**void** **increase**() {

**int** j = y;

**int** h = cy[j - m] - fy[j - m];

**while** (**true**) {

**int** i = p[j];

**if** (i == x) {

h = min(h, cx[i] - fx[i]);

**break**;

}

j = p[i];

h = min(h, f[i][j - m]);

}

j = y;

fy[j - m] += h;

**while** (**true**) {

**int** i = p[j];

f[i][j - m] += h;

**if** (i == x) {

fx[i] += h;

**break**;

}

j = p[i];

f[i][j - m] -= h;

}

}

**void** **repair**() {

**int** d = INF;

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

**if** (p[i] != -1)

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

**if** (p[j + m] == -1)

d = min(d, a[i][j] - gx[i] - gy[j]);

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

**if** (p[i] != -1) gx[i] += d;

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

**if** (p[j + m] != -1) gy[j] -= d;

}

**void** **solve**() {

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

**for** (**int** i = 0; i < m; i++) **scanf**("%d", &cx[i]), fx[i] = 0, gx[i] = INF;

**for** (**int** j = 0; j < n; j++) **scanf**("%d", &cy[j]), fy[j] = 0, gy[j] = 0;

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

**for** (**int** j = 0; j < n; j++) **scanf**("%d", &a[i][j]), f[i][j] = 0, gx[i] = min(gx[i], a[i][j]);

**for** (x = 0; x < m; x++)

**while** (fx[x] < cx[x]) {

**while** (!find())

repair();

increase();

}

ret = 0;

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

**for** (**int** j = 0; j < n; j++) ret += f[i][j] \* a[i][j];

**printf**("%d\n", ret);

}

## Rent a car

There are C motor companies in town, where the **kth** company has **ck** cars in their showroom and price of a car of this company is **pk**. There are **R** car service-centers in town, the **ith** center takes **di** days and costs **si** per car service. Service centers can service huge number of cars at the same time.

Now, ACM company has the request sheet for next **N** days, where in **jth** day, **rj** cars are needed. They want to fulfill all the requirements with minimized cost.

**void** **doit**() {

ll sum = 0;

For(i, 1, nday) sum += need[i];

**int** n = 2 \* nday + 2, s = 0, t = n - 1;

mcf.init(n, s, t);

For(i, 1, ncom) mcf.add(s, 1, has[i], price[i]);

For(i, 1, nday) mcf.add(s, nday + i, need[i], 0);

For(i, 1, nday) mcf.add(i, t, need[i], 0);

For(i, 1, nday - 1) mcf.add(i, i + 1, inf, 0);

For(i, 1, nser) For(j, 1, nday - take[i] - 1) mcf.add(nday + j, j + take[i] + 1, inf, cost[i]);

**if** (mcf.maxflow(sum)) **printf**("%lld\n", mcf.totalCost);

**else** **puts**("impossible");

}

## Similar Trees

**Kiểm tra 1 cây có phải là cây con của cây kia không.**

**struct** Tree {

**int** n, E, adj[maxe], next[maxe], last[maxv];

**void** **init**(**int** \_n) {

E = 0; n = \_n;

ms(last, -1);

}

**void** **add**(**int** u, **int** v) {

adj[E] = v; next[E] = last[u]; last[u] = E++;

adj[E] = u; next[E] = last[v]; last[v] = E++;

}

**bool** **isSubtreeOf**(Tree &other, **int** r1, **int** r2, **int** dad1 = -1, **int** dad2 = -1) {

**int** nchild1 = 0, nchild2 = 0;

**for** (**int** e1 = last[r1]; e1 != -1; e1 = next[e1]) **if** (adj[e1] != dad1) ++nchild1;

**for** (**int** e2 = other.last[r2]; e2 != -1; e2 = other.next[e2]) **if** (other.adj[e2] != dad2) ++nchild2;

**if** (nchild1 > nchild2) **return** **false**;

**if** (nchild1 == 0) **return** **true**;

**bool** graph[maxv][maxv];

ms(graph, **false**);

**for** (**int** e1 = last[r1]; e1 != -1; e1 = next[e1]) {

**int** u = adj[e1]; **if** (u == dad1) **continue**;

**for** (**int** e2 = other.last[r2]; e2 != -1; e2 = other.next[e2]) {

**int** v = other.adj[e2]; **if** (v == dad2) **continue**;

**if** (isSubtreeOf(other, u, v, r1, r2)) graph[u][v] = **true**;

}

}

hopkarp.init(n, other.n);

**for** (**int** e1 = last[r1]; e1 != -1; e1 = next[e1]) **if** (adj[e1] != dad1)

**for** (**int** e2 = other.last[r2]; e2 != -1; e2 = other.next[e2]) **if** (other.adj[e2] != dad2)

**if** (graph[adj[e1]][other.adj[e2]]) hopkarp.add(adj[e1], other.adj[e2]);

**return** hopkarp.maxmat() == nchild1;

}

**bool** **isSubtreeOf**(Tree &other) {

For(root, 1, other.n) **if** (isSubtreeOf(other, 1, root)) **return** **true**;

**return** **false**;

}

};

## Multiple Free Subset

Cho một tập số nguyên. Chọn ra một tập gồm nhiều số nhất sao cho không số nào là bội của số nào. (In ra tập có thứ tự từ điển bé nhất).

**inline** **bool** **found**() {

**memset**(trace, 0, **sizeof**(trace));

**int** first = 1, last = 0;

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

**if** (matchx[i] == 0) que[++last] = i;

**while** (first <= last) {

**int** i = que[first++];

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

**if** (trace[j] == 0 && rel[i][j]) {

trace[j] = i;

**if** (matchy[j] == 0) {

finish = j;

**return** **true**;

}

que[++last] = matchy[j];

}

}

**return** **false**;

}

**inline** **void** **enlarge**() {

++ ret;

**do** {

**int** x = trace[finish],

next = matchx[x];

matchx[x] = finish;

matchy[finish] = x;

finish = next;

} **while** (finish != 0);

}

**int** **main**() {

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

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

**scanf**("%d", a+i);

// 1. Erase equal numbers

sort(a + 1, a + n + 1);

**int** p = 1;

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

**if** (a[i] != a[i-1]) a[++p] = a[i];

n = p;

// 2. Creat bipartite graph

**memset**(rel, **false**, **sizeof**(rel));

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

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

rel[i][j] = (a[i] % a[j] == 0);

// 3. Matching to find minimum chains

ret = 0;

**memset**(matchx, 0, **sizeof**(matchx));

**memset**(matchy, 0, **sizeof**(matchy));

// Greedy

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

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

**if** (rel[i][j] && !matchx[i] && !matchy[j]) {

matchx[i] = j;

matchy[j] = i;

++ ret;

}

// Match

**while** (found()) enlarge();

ret = n - ret;

// Find independent set

**bool** Y[maxn+1], X[maxn+1];

**memset**(Y, **false**, **sizeof**(Y));

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

**if** (trace[i] != 0) Y[i] = **true**;

**memset**(X, **false**, **sizeof**(X));

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

**if** (matchx[i] != 0 && Y[ matchx[i] ] == **false**) X[i] = **true**;

**int** m = 0;

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

**if** (X[i] == **false** && Y[i] == **false**) ind[++m] = a[i];

**for** (**int** i = 1; i <= ret; ++i)

**printf**("%d%c", ind[i], i == ret ? '\n' : ' ');

}

## Tìm chữ số tận cùng khác 0 của N!

**const** **int** Tab[4][5][5] = {

{ {0,6,2,6,4},{2,2,4,2,8},{4,4,8,4,6},{6,6,2,6,4},{8,8,6,8,2} },

{ {0,2,8,8,4},{2,4,6,6,8},{4,8,2,2,6},{6,2,8,8,4},{8,6,4,4,2} },

{ {0,4,2,4,4},{2,8,4,8,8},{4,6,8,6,6},{6,4,2,4,4},{8,2,6,2,2} },

{ {0,8,8,2,4},{2,6,6,4,8},{4,2,2,8,6},{6,8,8,2,4},{8,4,4,6,2} }

};

**#define** MAXD 10000

**char** S[MAXD];

**int** A[MAXD], R[MAXD], N, NR, Rem, Ret, i;

**int** **main**() {

**while** (**scanf**("%s", S) == 1) {

**for** (N = **strlen**(S), i = 0; i < N; ++i) A[i+1] = S[N-i-1]-'0';

**if** (N == 1 && (A[1] == 0 || A[1] == 1)) Ret = 1; **else** {

**for** (NR = Rem = 0; NR < N;) {

++NR; R[NR] = A[NR] % 5;

**for** (i = NR; i <= N; ++i) {

A[i] = (A[i] << 1) + Rem;

Rem = A[i] / 10;

A[i] = A[i] % 10;

}

**if** (Rem == 1) {

A[++N] = 1;

Rem = 0;

}

A[NR] = 0;

}

**for** (Ret = 0, i = NR; i > 0; --i)

Ret = Tab[(i-1)%4][Ret >> 1][R[i]];

}

**printf**("%d\n", Ret);

}

**return** 0;

}

## Finding Fraction

Given a,b,c,d, find a fraction p/q with minimum q, and satisfied a/b < p/q < c/d.

**static** **int**[] run(**int** a, **int** b, **int** c, **int** d) {

**if** (a < b && c > d) **return** **new** **int**[] {1, 1};

**if** (a < b && c <= d) {

**int**[] ret = run(d, c, b, a);

**int** tmp = ret[0];

ret[0] = ret[1];

ret[1] = tmp;

**return** ret;

}

**int** diff = a/b;

a -= diff \* b;

c -= diff \* d;

**int**[] ret = run(a, b, c, d);

ret[0] += diff \* ret[1];

**return** ret;

}

## Tính LCM(1, 2, .., n) với n ≤ 108 (10000 test)

**const** **int** N = 100000000;

**const** **int** L = 1000;

bitset <N+1> prime;

vector <**int**> p;

vector <**unsigned** **int**> a;

**int** **where**(**int** n) {

**int** l = 0, r = p.size() - 1, m, ret = -1;

**while** (l <= r) {

m = (l + r) >> 1;

**if** (p[m] <= n) {

ret = m;

l = m + 1;

}

**else** r = m - 1;

}

**return** ret;

}

**unsigned** **int** **prod**(**int** i, **int** j) {

**unsigned** **int** ret = 1;

**int** k;

**for** (k = i; k <= j && k % L != 0; ++k) ret \*= p[k];

**for** (; k + L - 1 <= j; k += L) ret \*= a[k / L];

**for** (; k <= j; ++k) ret \*= p[k];

**return** ret;

}

**unsigned** **int** **lcm**(**int** n) {

**unsigned** **int** ret = 1;

**int** i;

**for** (i = 0; i < p.size() && p[i] <= n / p[i]; ++i) {

**int** x = 1;

**while** (x <= n / p[i]) x \*= p[i];

ret \*= x;

}

**if** (i < p.size()) {

**int** j = where(n);

ret \*= prod(i, j);

}

**return** ret;

}

**void** **init**() {

prime.set();

**for** (**int** i = 2; i <= N / i; ++i)

**if** (prime.test(i))

**for** (**int** j = i \* i; j <= N; j += i) prime.reset(j);

**for** (**int** i = 2; i <= N; ++i)

**if** (prime.test(i)) p.push\_back(i);

**unsigned** **int** product = 1;

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

product \*= p[i];

**if** ((i + 1) % L == 0) {

a.push\_back(product);

product = 1;

}

}

}

## Đếm số cách đặt K quân tượng lên bàn cờ N\*N sao cho không ăn nhau

**int** **cells**(**int** i) {

**if** (i & 1) **return** i / 4 \* 2 + 1;

**else** **return** (i - 1) / 4 \* 2 + 2;

}

**int** **bishop**(**int** n, **int** k) {

**if** (k > 2 \* n - 1) **return** 0;

vector<vector<**int**> > d(n \* 2, vector<**int**>(k + 2));

**for** (**int** i = 0; i < n \* 2; ++i) d[i][0] = 1;

d[1][1] = 1;

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

**for** (**int** j = 1; j <= k; ++j)

d[i][j] = d[i - 2][j] + d[i - 2][j - 1] \* (cells(i) - j + 1);

**int** ans = 0;

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

ans += d[n \* 2 - 1][i] \* d[n \* 2 - 2][k - i];

**return** ans;

}

## Black and White Nim

Có một hoặc nhiều hàng, mỗi hàng chứa một số hạt màu đen và trắng. Hai người chơi lần lượt thực hiện lượt đi bằng cách bỏ đi các hạt cho đến hết. Trong mỗi lượt đi, một người chơi phải bỏ đi một hoặc nhiều hạt liên tiếp từ đầu bên trái của một hàng. Các hạt bị bỏ đi phải chứa hoặc là không hạt nào màu đen, hoặc là có duy nhất một hạt đen. Nếu một hạt màu đen bị bỏ đi, hạt đó phải là hạt bên phải nhất của dãy các hạt bị bỏ. Người nào đi lượt cuối cùng là người chiến thắng.

Cho số lượng hạt màu đen và màu trắng ở mỗi hàng. Thứ tự của các hạt trên một hàng được tạo ra một cách ngẫu nhiên vào thời điểm bắt đầu trò chơi. Thứ tự của các hàng phân biệt có thể giống nhau. Các hạt màu đen giống hệt nhau, và các hạt màu trắng cũng giống hệt nhau (không thể phân biệt các hạt cùng màu). In ra xác suất mà người chơi đầu tiên giành chiến thắng nếu như cả hai người đều chơi tối ưu.

**double** f[102][102][128], g[102][102][128], d[111][128];

**int** test, n, a[111], b[111];

**int** **main**() {

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

**memset**(g, 0, **sizeof**(g));

**for** (**int** i = 0; i <= 100; i++)

**for** (**int** j = 0; j <= 100; j++) {

**if** (i == 0 && j == 0) {

f[i][j][0] = 1;

} **else** **if** (i == 0) {

**if** (j & 1)

f[i][j][1] = 1;

**else**

f[i][j][0] = 1;

} **else** **if** (j == 0) {

f[i][j][i] = 1;

} **else** {

**double** r = j \* 1. / (i + j);

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

f[i][j][k + 1] += r \* g[i - k][j - 1][k];

f[i][j][k] += r \* (1 - g[i - k][j - 1][k]);

**if** (k < i)

r \*= (i - k) \* 1. / (i + j - k - 1);

}

}

g[i][j][0] = f[i][j][0];

**for** (**int** k = 1; k < 128; k++)

g[i][j][k] = g[i][j][k - 1] + f[i][j][k];

}

**scanf**("%d", &test);

**for** (**int** itest = 0; itest < test; itest++) {

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

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

**scanf**("%d", &a[i]);

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

**scanf**("%d", &b[i]);

**memset**(d, 0, **sizeof**(d));

d[0][0] = 1;

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

**for** (**int** j = 0; j < 128; j++) {

**for** (**int** k = 0; k < 128; k++) {

d[i][j] += d[i - 1][k] \* f[b[i]][a[i]][j ^ k];

}

}

}

**printf**("%.6lf\n", 1 - d[n][0]);

}

}

## HUGEKNAP

**#define** maxn 10000

**struct** Gold {

**int** idx;

ll w, v;

**bool** **operator <** (**const** Gold& that) **const** {

**return** v \* that.w > w \* that.v;

}

};

**int** ntest, n, s[maxn + 5], r[maxn + 5];

ll wmax, vnow, vmax, wnow;

Gold a[maxn + 5];

ll **bound**(**int** i) {

ll w = wnow + a[i].w, v = vnow + a[i].v;

**while** (w <= wmax) { ++ i; w += a[i].w; v += a[i].v; }

w -= a[i].w; v -= a[i].v;

**return** v + (wmax - w) \* a[i].v / a[i].w;

}

**void** **go**(**int** pos) {

For (i, s[pos - 1] + 1, n)

**if** (wnow + a[i].w <= wmax) {

**if** (bound(i) <= vmax) **return**;

wnow += a[i].w; vnow += a[i].v; s[pos] = i;

**if** (vmax < vnow) {

vmax = vnow;

r[0] = pos;

For (j, 1, pos) r[j] = a[ s[j] ].idx;

}

**if** (pos < n) go(pos + 1);

wnow -= a[i].w; vnow -= a[i].v; s[pos] = 0;

}

}

**int** **main**() {

**scanf**("%d %lld", &n, &wmax);

For (i, 1, n) a[i].idx = i;

For (i, 1, n) **scanf**("%lld", &a[i].w);

For (i, 1, n) **scanf**("%lld", &a[i].v);

a[n + 1].idx = n + 1; a[n + 1].w = wmax + 1; a[n + 1].v = 0;

sort(a + 1, a + n + 1);

wnow = vnow = vmax = 0;

s[0] = 0; go(1);

**printf**("%lld %d\n", vmax, r[0]);

For (i, 1, r[0]) **printf**("%d ", r[i]);

}

GOODLUCK!