The Team Reference Document of UWr1

Supported by: prof. dr hab. Krzysztof Loryś, dr Paweł Gawrychowski, mgr Bartłomiej Dudek

Spis treści

File's control sum:

```
cat file | tr -d "[:space:]" | md5sum
```

1. Vimrc

ff14ab2e0f8ebb23b8d91f9f9ab3582f

```
syn on
set nu cin sw=4 ts=4 so=999
no <F4> :w <bar> :!make %:r && ./%:r <cr>
```

2. Template

afe54fba633b5365328211e95ab4235b

```
#pragma GCC optimize ("O3")
#include <bits/stdc++.h>
using namespace std;
#define rep(i, a, b) for (int i = (a); i <= (b); i++)
#define per(i, a, b) for (int i = (b); i >= (a); i--)
#define SZ(x) ((int)x.size())
#define all(x) x.begin(),x.end()
#define pb push_back
#define mp make_pair
#define st first
#define nd second
using ll = long long;
using lf = long double;
using pii = pair <int, int>;
using pll = pair <ll, ll>;
auto& operator<<(auto& o, pair <auto, auto> p){
    return o << "(" << p.first << ", " << p.second << ")";
}
auto operator<<(auto& o, auto x)->decltype(end(x), o){
    o << "{"; int i = 0; for (auto e : x) o << ", " + 2*!i++ << e;
    return o << "}";
}
#define dbg(x...) cerr<<"["#x"]: ",[](auto...$){((cerr<<$<<"; "),...)<<endl;}(x)
```

3. Pragmas

7cabee7e3329d18592ba8797834694c4

```
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,avx,avx2")
#pragma GCC target("popcnt,abm,mmx,avx,tune=native")
```

4. Makefile

1c3262b6ed1e3d4de3058ef10b313146

```
CXXFLAGS = -Wall -Wextra -Wshadow -std=c++2a -fsanitize=address -fsanitize=undefined
```

5. 2SAT

8363b0e12f36c79f8eeaca6b23b98816

```
struct twosat { /* 0-indexed */
    int n, cnt; vector <vector <int> > G, R;
    vector <int> order, comp, ans; vector <bool> vis;
    twosat(int _n) : n(_n) {
        G.resize(n + n); R.resize(n + n); comp.resize(n + n); vis.resize(n + n);
        ans.resize(n); order.reserve(n + n);
    }
    void add_edge(int u, int v) { G[u].push_back(v); R[v].push_back(u); }
    void add_clause(int u, bool fu, int v, bool fv) {
        add_edge(u << 1 | !fu, v << 1 | fv); add_edge(v << 1 | !fv, u << 1 | fu);
    }
    void dfs(int u) {
        vis[u] = true;
        for (const auto &v: G[u]) if (!vis[v]) dfs(v);
        order.push_back(u);
    }
    void scc(int u, int id) {
        vis[u] = true, comp[u] = id;
        for (const auto &v: R[u]) if (!vis[v]) scc(v, id);
    }
    bool run() {
        for (int i = 0; i < n + n; ++i) if (!vis[i]) dfs(i);
        fill(vis.begin(), vis.end(), false);
        reverse(order.begin(), order.end());
        for (const auto &v: order) if (!vis[v]) scc(v, ++cnt);
        for (int i = 0; i < n; ++i) {
            if (comp[i << 1] == comp[i << 1 | 1])   return false;
            ans[i] = comp[i << 1] < comp[i << 1 | 1];
        }
        return true;
    }
};
```

6. Dinic

ee279b38c7ec82ffad2853c69d09fc8a

```
#define sz(x) (int)(x).size()
typedef int T;
const int N = 2005;
const T INF = 1e9;
struct edge { int a, b; T cap, flow; };
int n, s, t, d[N], ptr[N], q[N];
vector < edge > e;
vector < int > g[N];
void clear() {
        e.clear();
        for(int i = 1; i <= n; ++i) g[i].clear();
} /* Assumes all edges added with [add_edge] */
void clear_flow() {
        for(int i = 0; i < (int)e.size(); i += 2)
                e[i].flow = 0, e[i + 1].flow = e[i + 1].cap;
} /* The only way to add edges */
int add_edge(int a, int b, T cap) {
    edge e1 = { a, b, cap, 0 };
    edge e2 = { b, a, cap, cap };
    g[a].push_back(sz(e)); e.push_back(e1);
    g[b].push_back(sz(e)); e.push_back(e2);
    return sz(e) - 2;
```

```
}
bool bfs() {
    int qh = 0, qt = 0; q[qt++] = s;
    memset(d + 1, -1, n * sizeof d[0]); d[s] = 0;
    while(qh < qt && d[t] == -1) {
        int v = q[qh++];
        for(int i = 0; i < sz(g[v]); ++i) {
            int id = g[v][i], to = e[id].b;
            if(d[to] == -1 && e[id].flow < e[id].cap) {
                q[qt++] = to; d[to] = d[v] + 1;
            }
        }
    }
    return d[t] != -1;
}
T dfs(int v, T flow) {
    if(flow <= 0)  return 0;
    if(v == t)  return flow;
    T res = 0;
    for(; ptr[v]<sz(g[v]); ++ptr[v]) {
        int id = g[v][ptr[v]], to = e[id].b;
        if(d[to] != d[v] + 1) continue;
        T pushed = dfs(to, min(flow, e[id].cap - e[id].flow));
        e[id].flow += pushed;
        e[id^1].flow -= pushed;
        res += pushed;
        flow -= pushed;
        if(flow == 0) break;
    }
    return res;
} /* 1-indexed */
T dinic(int _n, int _s, int _t) {
    n = _n; s = _s; t = _t; T flow = 0;
    for (;;) {
        if(!bfs())  break;
        memset(ptr, 0, (n + 1) * sizeof ptr[0]);
        flow += dfs(s, INF);
    }
    return flow;
}
```

7. Gomory-Hu

6bfc4a85537b9dd16bbd0dc039a29c4b

```
struct edge { int u, v; long long w; };
int n; vector <int> p, w, c; vector <edge> tree;
void dfs(int u) {
        c[u] = 1;
        for(const int &id: Dinic::g[u]) {
                int v = Dinic::e[id].b;
                if(!c[v] and Dinic::e[id].flow < Dinic::e[id].cap) dfs(v);
        }
} /* Clears and runs */
vector <edge> run(int _n, const vector <edge> &ed) {
        n = _n; tree.clear(); p.resize(n + 1), w.resize(n + 1), c.resize(n + 1);
        for(const auto &e: ed) {
                Dinic::add_edge(e.u, e.v, e.w); Dinic::add_edge(e.v, e.u, e.w);
        }
        p[1] = 0, fill(p.begin() + 2, p.end(), 1);
        for(int i = 2; i <= n; ++i) {
                w[i] = Dinic::dinic(n, i, p[i]);
                fill(c.begin(), c.end(), 0); dfs(i);
                for(int j = i + 1; j <= n; ++j) if(c[j] && p[j] == p[i]) p[j] = i;
                if(p[p[i]] && c[p[p[i]]]) {
                        int pi = p[i]; swap(w[i], w[pi]); p[i] = p[pi]; p[pi] = i;
                }
                Dinic::clear_flow();
        }
        for(int i = 1; i <= n; ++i) { if(p[i]) tree.push_back(edge{i, p[i], w[i]}); }
        return tree;
}
```

8. Min Cost Max Flow

72f01e19eb4ad784924072ba550105ec

```
typedef int flow_t; typedef int cost_t;
struct edge {
    int u, v;
    flow_t flow, capa; cost_t cost; };
const int N = 10000;
const cost_t cinf = 1e9;
const flow_t finf = 1e9;
vector<int> g[N];
cost_t d[N],p[N];
int n, pre[N];
vector<edge> e;
inline bool remin(cost_t &a, cost_t b) {
    return a > b ? a = b, true : false;
}//wierzcholki numerowane od 0 do n - 1
void init(int _n) {
    n = _n; e.clear();
    for(int i = 0; i < n; i++) g[i].clear();
}
void add_edge(int u, int v, flow_t capa, cost_t cost) {
    g[u].push_back(e.size());
    e.push_back({ u, v, 0, capa, cost });
    g[v].push_back(e.size());
    e.push_back({ v, u, 0, 0, -cost });
}
pair<flow_t, cost_t> flow(int s, int t) {
    fill(p, p + n, 0);
    bool improved = true;
    while(improved) {
        improved = false;
        for(auto &ed: e)
            if(ed.flow < ed.capa && remin(p[ed.v], p[ed.u] + ed.cost))
                improved = true;
    }
    flow_t fans = 0; cost_t cans = 0;
    while(true) {
        fill(d, d + n, cinf);
        priority_queue<pair<cost_t, int>> q;
        d[s] = 0; q.push({ 0, s });
        while(!q.empty()) {
            auto u = q.top().second, c = -q.top().first;
            q.pop();
```

```
            if(c != d[u]) continue;
            for(int ed: g[u]) {
                if(e[ed].flow == e[ed].capa) continue;
                auto v = e[ed].v;
                if(remin(d[v], c + p[u] - p[v] + e[ed].cost)) {
                    pre[v] = ed;q.push({ -d[v], v });
                }
            }
        } if(d[t] == cinf) break;
        vector<int> path;
        int v = t, ed = pre[v];
        flow_t flow = finf;
        while(v != s) {
            path.push_back(ed);
            flow = min(flow, e[ed].capa - e[ed].flow);
            v = e[ed].u; ed = pre[v];
        }
        for(auto ed: path) {
            e[ed].flow += flow;
            e[ed^1].flow -= flow;
        }
        fans += flow;
        cans += flow * (d[t] + p[t] - p[s]);
        for(int i = 0; i < n; i++) p[i] += d[i];
    }
    return { fans, cans };
}
```

9. Hungarian

bf99fa2858cd62d5c725aa617899112d

```
typedef int T; const int N = 507; const T INF = 1e9 + 7;
int n, max_match;
T cost[N][N], lx[N], ly[N], slack[N], slackx[N];
int xy[N], yx[N], prev[N]; bool S[N], U[N];
void update_labels() {
    T delta = INF;
    FOR(y, n) if(!U[y]) delta = min(delta, slack[y]);
    FOR(x, n) if(S[x]) lx[x] -= delta;
    FOR(y, n) if(U[y]) ly[y] += delta;
    FOR(y, n) if(!U[y]) slack[y] -= delta;
}
void add_to_tree(int x, int f) {
    S[x] = true; prev[x] = f;
    FOR(y, n) if(lx[x] + ly[y] - cost[x][y] < slack[y]) {
        slack[y] = lx[x] + ly[y] - cost[x][y]; slackx[y] = x;
    }
}
void augment() {
    if (max_match == n) return;
    int root, q[N], wr = 0, rd = 0;
    memset(S, false, sizeof(S)); memset(U, false, sizeof(U));
    memset(prev, -1, sizeof(prev));
    FOR(x, n) if(xy[x] == -1) {
        q[wr++] = root = x; prev[x] = -2; S[x] = true; break;
    }
    FOR(y, n) { slack[y] = lx[root] + ly[y] - cost[root][y]; slackx[y] = root; }
    int x, y;
    while(true) {
        while(rd < wr) {
            x = q[rd++];
            for(y = 0; y < n; y++) {
                if(cost[x][y] == lx[x] + ly[y] && !U[y]) {
                    if (yx[y] == -1) break;
                    U[y] = true; q[wr++] = yx[y]; add_to_tree(yx[y], x);
                }
            }
            if(y < n) break;
        }
        if (y < n) break;
        update_labels(); wr = rd = 0;
        for(y = 0; y < n; y++) {
            if(!U[y] && slack[y] == 0) {
                if(yx[y] == -1) { x = slackx[y]; break; }
                else {
                    U[y] = true;
                    if(!S[yx[y]]) {
                        q[wr++] = yx[y]; add_to_tree(yx[y], slackx[y]);
                    }
                }
            }
        }
        if(y < n) break;
    }
    if(y < n) {
        max_match++;
        for(int cx = x, cy = y, ty; cx != -2; cx = prev[cx], cy = ty) {
            ty = xy[cx]; yx[cy] = cx; xy[cx] = cy;
        }
        augment();
    }
}
T maxCostMatching() {
    T res = 0; max_match = 0;
    FOR(i, n) xy[i] = yx[i] = -1, lx[i] = ly[i] = 0;
    FOR(x, n) FOR(y, n) lx[x] = max(lx[x], cost[x][y]);
    augment(); FOR(x, n) res += cost[x][xy[x]];
    return res;
}
```

10. General Matching

284bc8de1ad931bf086fa111059dcfe1

```
mt19937 rnd(time(0)); /* [1, n]*/
vector<int> e[N]; int mate[N], vis[N];
void add(int a, int b) {
    if (a != b) { e[a].push_back(b); e[b].push_back(a); }
}
bool dfs(int a) {
    shuffle(e[a].begin(), e[a].end(), rnd); vis[a] = 1;
    for (auto b : e[a]) {
        int c = mate[b]; if (vis[c]) continue;
        mate[a] = b; mate[b] = a; mate[c] = 0;
        if (!c || dfs(c)) return 1;
        mate[a] = 0; mate[b] = c; mate[c] = b;
    }
```

```
    return 0;
}
vector<pair<int, int>> matching(int n) {
    vector<pair<int, int>> res;
    rep(_, 1, 20) {
        memset(mate, 0, sizeof mate);
        rep(i, 1, n) {
            if (!mate[i]) { memset(vis, 0, sizeof vis); dfs(i); }
        }
        vector<pair<int, int>> cur;
        rep(i, 1, n) if (mate[i] > i) cur.push_back({i, mate[i]});
        if (cur.size() > res.size()) res = cur;
    }
    return res;
}
```

11. Weighted Graph Matching

357c8dc9abf9d8dce960a34b136c0e1b

```
long long G[N][N], dis[N];
int n, top, match[N], mat[N], stk[N], id[N], vis[N];
const long long inf = 1e18;
void init(int _n) {
    n = _n; top = 0; memset(match, 0, sizeof match);
    for (int i = 1; i <= n + 1; i++)
        for (int j = 1; j <= n + 1; j++) G[i][j] = 0;
}
void add_edge(int u, int v, long long w) {
    G[u][v] = max(G[u][v], w); G[v][u] = max(G[v][u], w);
}
bool spfa(int u) {
    stk[top ++] = u;
    if (vis[u]) return true;
    vis[u] = true;
    for (int i = 1; i <= n; ++ i) {
        if (i != u && i != mat[u] && !vis[i]) {
            int v = mat[i];
            if (dis[v] < dis[u] + G[u][i] - G[i][v]) {
                dis[v] = dis[u] + G[u][i] - G[i][v];
                if (spfa(v)) return true;
            }
        }
    }
    top--; vis[u] = false;
    return false;
}
long long maximum_matching() {
    for (int i = 1; i <= n; ++ i) id[i] = i;
    for (int i = 1; i <= n; i += 2) mat[i] = i + 1, mat[i + 1] = i;
    for (int times = 0, flag; times < 3; ) { /* more iters, better prob */
        memset(dis, 0, sizeof(dis)); memset(vis, 0, sizeof(vis));
        top = 0; flag = 0;
        for (int i = 1; i <= n; ++ i) {
            if (spfa(id[i])) {
                flag = 1; int t = mat[stk[top - 1]], j = top - 2;
                while (stk[j] != stk[top - 1]) {
                    mat[t] = stk[j]; swap(t, mat[stk[j]]); --j;
                }
                mat[t] = stk[j]; mat[stk[j]] = t; break;
            }
        }
        if (!flag) { times ++; random_shuffle(id + 1, id + n + 1); }
    }
    long long ans = 0;
    for (int i = 1; i <= n; ++ i) {
        if (mat[i] <= n && i < mat[i]) {
            if (G[i][mat[i]] != 0)
                ans += G[i][mat[i]], match[i] = mat[i], match[mat[i]] = i;
        }
    }
    return ans;
}
```

12. Dominator Tree

f60f3bb9d21ed5a7ff0366b7cbb7bb5c

```
struct dominator_tree { /* 0 indexed */
    vector<basic_string<int>> g, rg, bucket;
    basic_string<int> arr, par, rev, sdom, dom, dsu, label;
    int n, t;
    dominator_tree(int n) : g(n), rg(n), bucket(n), arr(n, -1),
          par(n, -1), rev(n, -1), sdom(n, -1), dom(n, -1),
          dsu(n, 0), label(n, 0), n(n), t(0) {}
    void add_edge(int u, int v) { g[u] += v; }
    void dfs(int u) {
        arr[u] = t; rev[t] = u;
        label[t] = sdom[t] = dsu[t] = t; t++;
        for (int w : g[u]) {
            if (arr[w] == -1) {
                dfs(w);
                par[arr[w]] = arr[u];
            }
            rg[arr[w]] += arr[u];
        }
    }
    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];
    } /* returns -1 for unreachable */
    vector<int> run(int root) {
        dfs(root);
        iota(dom.begin(), dom.end(), 0);
        for (int i = t - 1; i >= 0; i--) {
            for (int w : rg[i]) sdom[i] = min(sdom[i], sdom[find(w)]);
            if (i) bucket[sdom[i]] += i;
            for (int w : bucket[i]) {
                int v = find(w);
                if (sdom[v] == sdom[w])
                    dom[w] = sdom[w];
                else
                    dom[w] = v;
            }
```

```
            if (i > 1) dsu[i] = par[i];
        }
        for (int i = 1; i < t; i++) {
            if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
        }
        vector<int> outside_dom(n, -1);
        for (int i = 1; i < t; i++) outside_dom[rev[i]] = rev[dom[i]];
        return outside_dom;
    }
};
```

13. Directed MST

ebb02b3ab6583bb66d9998e92f93c675

```
// directed mst z wierzcholka 0 w grafie 0..n-1
// preconditions: 1. nie ma krawedzi wchodzacych do wierzcholka 0
//                2. wszystkie wierzcholki sa osiagalne z wierzcholka 0
// przed uzyciem ustawic n, m, MAXN, MAXM i wypelnic
//edge[0..m-1] (wystarczy pola: u, v, key)
const int MAXN = 100007, MAXM = 100007;
struct edge { // krawedz/element kolejki zlaczalnej
    int u, v; // IN: poczatek i koniec krawedzi
    int key;  // IN: waga krawedzi (zmienia sie!)
    edge *left, *right; // poczatkowo: 0, 0
    int len, add;       // poczatkowo: 1, 0
};
struct node1 { // element zbioru
    node1 *parent; int size, scc;
};
struct node2 { // j.w.
    node2 *parent; // poczatkowa wartosc: this
    int size;      // poczatkowa wartosc: 1
};
// Operacje na zbiorach rozlacznych
template<class T> T *set_find(T *p) { // znajduje reprezentanta
    if (p->parent != p) p->parent = set_find(p->parent);
    return p->parent;
}
template<class T> T *set_union(T *p1, T *p2) { // laczy zbiory
    if (p1->size < p2->size) swap(p1, p2);
    p2->parent = p1;
    p1->size += p2->size;
    return p1;
}
// Operacje na kolejkach zlaczalnych
void tree_push(edge *p) {
    p->key += p->add;
    if (p->left) p->left->add += p->add;
    if (p->right) p->right->add += p->add;
    p->add = 0;
}
edge *tree_union(edge *p1, edge *p2) { // laczy kolejki
    if (!p1) return p2; if (!p2) return p1;
    if (p2->key+p2->add < p1->key+p1->add) swap(p1, p2);
    tree_push(p1);
    p1->right = tree_union(p1->right, p2);
    if (!p1->left || p1->left->len < p1->right->len) swap(p1->left, p1->right);
    p1->len = p1->right ? p1->right->len+1 : 1;
    return p1;
}
edge *tree_extract(edge *p) { // usuwa z kolejki element najmniejszy
    tree_push(p); return tree_union(p->left, p->right);
}
void tree_add(edge *p, int x) { // dodaje x do wszystkich wartosci w kolejce
    if (p) p->add += x;
}
int n, m;          // IN: liczba wierzcholkow, liczba krawedzi
edge edges[MAXM]; // IN: tablica wszystkich krawedzi
node1 scc_set[MAXN];
node2 wcc_set[MAXN];
int upper[2*MAXN], lower[2*MAXN];
edge *adj[2*MAXN];
edge *res[2*MAXN]; // OUT: krawedz do rodzica w drzewie (korzen ma NULL)
int compute_branching() { // zwraca wage drzewa
    FOR(i,n) {
        scc_set[i].parent = scc_set+i;
        scc_set[i].size = 1;
        scc_set[i].scc = i;
        wcc_set[i].parent = wcc_set+i;
        wcc_set[i].size = 1;
        upper[i] = lower[i] = -1;
        adj[i] = res[i] = 0;
    }
    FOR(j,m) {
        edges[j].left = edges[j].right = 0;
        edges[j].len = 1;
        edges[j].add = 0;
        adj[edges[j].v] = tree_union(adj[edges[j].v], edges+j);
    }
    int scc_c=n, value=0;
    FOR(i,n) {
        int c = set_find(scc_set+i)->scc;
        while (adj[c] && !res[c]) {
            edge *e = adj[c];
            adj[c] = tree_extract(adj[c]);
            node1 *s1 = set_find(scc_set+e->v), *s2 = set_find(scc_set+e->u);
            if (s1==s2) continue;
            res[c] = e;
            value += e->key;
            tree_add(adj[c], -e->key);
            node2 *w1 = set_find(wcc_set+e->v), *w2 = set_find(wcc_set+e->u);
            if (w1!=w2) { set_union(w1, w2); continue; }
            upper[c] = scc_c;
            do {
                e = res[s2->scc];
                upper[s2->scc] = scc_c;
                adj[c] = tree_union(adj[c], adj[s2->scc]);
                s1 = set_union(s1, s2);
                s2 = set_find(scc_set+e->u);
            } while (s1!=s2);
            s1->scc = scc_c;
            upper[scc_c] = lower[scc_c] = -1;
            adj[scc_c] = adj[c];
            res[scc_c] = 0;
            c = scc_c++;
        }
    }
```

```
    REPD(c,scc_c - 1,n) {
        if (lower[c]==-1)
          for (int i=res[c]->v; i!=c; i=upper[i]) lower[upper[i]] = i;
        res[lower[c]] = res[c];
    }
    return value;
}
```

14. Turbo matching

565529bc2bd2546c12d5ac514d100004

```
struct Matching {
    int n; vector < int > *G, match, vis;
    bool dfs(int v) {
        vis[v] = 1;
        for(auto u: G[v]) if(!match[u] || (!vis[match[u]] && dfs(match[u]))) {
                match[v] = u; match[u] = v; return true;
        }
        return false;
    }
    Matching(int N, vector < int > *g) : n(N), G(g) {
        match.resize(n + 1, 0), vis.resize(n + 1, 0); bool ok = 1;
        while(ok) {
            ok = 0;
            for(int i = 1; i <= n; i++) if(!match[i] && dfs(i)) ok = 1;
            for(int i = 1; i <= n; i++) vis[i] = 0;
        }
    }
};
```

15. All maximal cliques

cce8d11672079488600745328360 5bb8

```
/* Description: Runs a callback for all maximal cliques in a graph (given as a
 * symmetric bitset matrix; self-edges not allowed). Callback is given a bitset
 * representing the maximal clique.
 * Time: O(3^{n/3}), much faster for sparse graphs
 */
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
        if (!P.any()) { if (!X.any()) f(R); return; }
        auto q = (P | X)._Find_first();
        auto cands = P & ~eds[q];
        FOR(i,0,SZ(eds)) if (cands[i]) {
                R[i] = 1;
                cliques(eds, f, P & eds[i], X & eds[i], R);
                R[i] = P[i] = 0; X[i] = 1;
        }
}
```

16. Policy-Based Data Structures

7af430e0eab65bafd07890a056a5f4a5

```
/* include <ext/pb_ds/assoc_container.hpp> and <ext/pb_ds/tree_policy.hpp>,
   namespace [__gnu_pbds] for multiset: less_equal<T>, for map: null_type change to T2
   [find_by_order] (int -> int*) and  [order_of_key] (int -> int) */
template <class T> using ordered_set =
    tree <T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
```

17. Fast hashtable

61735f813571a7a376aa53c841a7f932

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct chash {
        const uint64_t C = ll(4e18 * acos(0)) | 71;
        ll operator()(ll x) const { return __builtin_bswap64((x^1234567891)*C); }
};// 1 << 16 is initial size, size has to be the power of 2
gp_hash_table<ll, int, chash> h({},{},{},{}, {1 << 16});
```

18. Set comparator

55698d723e5131fcbb64936b846a3223

```
struct Comparator {
    bool operator()(const T a, const T b) const {
        return a < b;
    }
};
set <T, Comparator> S;
priority_queue<T, vector <T>, Comparator> Q;
```

19. Fenwick

6961bad25d49874985dd870e7676f291

```
template < typename TP >
struct Fenwick { // + / sum from 0 to n
        int N = 1e5 + 7; vector < TP > tree;
        Fenwick() { tree.resize(N + 2, 0); }
        Fenwick(int n) : N(n + 2) { tree.resize(N + 2, 0); }
        void update(int id, TP val) {
                for(int i = id + 1; i <= N + 1; i += (i & -i)) tree[i] += val;
        }
        TP pref(int x) {
                TP res = 0;
                for(int i = x + 1; i; i -= (i & -i)) res += tree[i];
                return res;
        }
        TP query(int l, int r) { return pref(r) - pref(l - 1); }
}; /* 2D Fenwick, */
int MAX_X = 1e5, MAX_Y = 1e5; /* map of pair is also fine */
unordered_map < int, unordered_map < int, LL > > tree;
inline void update(int x, int y, LL val) {
    for(int i = x + 1; i <= MAX_X + 1; i += (i & -i))
        for(int j = y + 1; j <= MAX_Y + 1; j += (j & -j)) tree[i][j] += val;
}
inline LL pref(int x, int y) {
    LL res = 0;
    for(int i = x + 1; i; i -= (i & -i)) {
        if(tree.find(i) == tree.end()) continue;
```

```
        for(int j = y + 1; j; j -= (j & -j))
            if(tree[i].find(j) != tree[i].end()) res += tree[i][j];
    }
    return res;
}
inline LL query(int x1, int x2, int y1, int y2) {
    return pref(x2, y2) - pref(x2, y1 - 1) - pref(x1 - 1, y2) + pref(x1 - 1, y1 - 1);
}
```

20. Link Cut Tree

3de26330f43cb16a3249de56e7802dbf

```
struct SplayTree {
  struct Node {
    int ch[2] = {0, 0}, p = 0;
    long long self = 0, path = 0; /* Path aggregates */
    long long sub = 0, vir = 0; /* subtree aggregates */
    bool flip = 0; /* lazy tags */
  };
  vector<Node> T;
  SplayTree(int n) : T(n + 1) {}
  void push(int x) {
    if (!x || !T[x].flip) return;
    int l = T[x].ch[0], r = T[x].ch[1];
    T[l].flip ^= 1, T[r].flip ^= 1;
    swap(T[x].ch[0], T[x].ch[1]);
    T[x].flip = 0;
  }
  void pull(int x) {
    int l = T[x].ch[0], r = T[x].ch[1]; push(l); push(r);
    T[x].path = T[l].path + T[x].self + T[r].path;
    T[x].sub = T[x].vir + T[l].sub + T[r].sub + T[x].self;
  }
  void set(int x, int d, int y) { T[x].ch[d] = y; T[y].p = x; pull(x); }
  void splay(int x) {
    auto dir = [&](int x) {
      int p = T[x].p; if (!p) return -1;
      return T[p].ch[0] == x ? 0 : T[p].ch[1] == x ? 1 : -1;
    };
    auto rotate = [&](int x) {
      int y = T[x].p, z = T[y].p, dx = dir(x), dy = dir(y);
      set(y, dx, T[x].ch[!dx]); set(x, !dx, y);
      if (~dy) set(z, dy, x); T[x].p = z;
    };
    for (push(x); ~dir(x); ) {
      int y = T[x].p, z = T[y].p;
      push(z); push(y); push(x);
      int dx = dir(x), dy = dir(y);
      if (~dy) rotate(dx != dy ? x : y); rotate(x);
    }
  }
};
struct LinkCut : SplayTree {
  LinkCut(int n) : SplayTree(n) {}
  int access(int x) {
    int u = x, v = 0;
    for (; u; v = u, u = T[u].p) {
      splay(u); int& ov = T[u].ch[1];
      T[u].vir += T[ov].sub; T[u].vir -= T[v].sub;
      ov = v; pull(u);
    }
    return splay(x), v;
  } /* be careful with rooted trees! */
  void reroot(int x) { access(x); T[x].flip ^= 1; push(x); }
  void Link(int u, int v) {   /* add u as child of v */
    reroot(u); access(v);
    T[v].vir += T[u].sub;
    T[u].p = v; pull(v);
  } /* remove edge [u, v] */
  void Cut(int u, int v) {
    reroot(u); access(v); T[v].ch[0] = T[u].p = 0; pull(v);
  } /* 0 if not connected */
  int LCA(int u, int v) {
    if (u == v) return u; access(u); int ret = access(v);
    return T[u].p ? ret : 0;
  } /* query u's subtree, v is outside of it */
  long long Subtree(int u, int v) {
    reroot(v); access(u); return T[u].vir + T[u].self;
  } /* query path [u..v] */
  long long Path(int u, int v) {
    reroot(u); access(v); return T[v].path;
  } /* Set value in vertex u to v */
  void Update(int u, long long v) { access(u); T[u].self = v; pull(u); }
};
```

21. Subset DP

af75ad8b948bebc93454866561dbed1f

```
vector <int> solve(int W, vector <int> coins) {
    int n = coins.size(), all_sum = 0;
    if (n == 0) { return 0; }
    vector <int> dp[2];
    for (auto v: coins) { all_sum += v; }
    int goal = all_sum / 2, start_idx = 0, cur_sum = 0;
    while (cur_sum + coins[start_idx] <= goal) {
        cur_sum += coins[start_idx++];
    }
    start_idx--;
    for (int i = 0; i < 2; ++i) { dp[i].assign(W + W + 1, -1); }
    auto update = [&](const int id) {
        for (int i = W + W; i >= 0; --i) {
            for (int j = max(0, dp[id ^ 1][i]); j < dp[id][i]; ++j) {
                if (i - coins[j] >= 0) {
                    dp[id][i - coins[j]] = max(dp[id][i - coins[j]], j);
                }
            }
        }
    };
    dp[start_idx & 1][cur_sum + W - goal] = start_idx + 1;
    update(start_idx & 1);
    for (int i = start_idx + 1; i < n; ++i) {
        const int id = i & 1;
        dp[id].assign(W + W + 1, -1);
        for (int j = 0; j + coins[i] <= W + W; ++j) {
            dp[id][j] = max(dp[id][j], dp[id ^ 1][j]);
            dp[id][j + coins[i]] = max(dp[id][j + coins[i]], dp[id ^ 1][j]);
```

```
        }
        update(id);
    }
    return dp[(n - 1) & 1];
}
```

22. Linear function max

9af308ba86b4c02f6119bfbc1f0eb402

```
struct line { // f(x) = ax + b
        ll a, b; mutable ll p;
        bool operator<(const line& o) const { return a < o.a; }
        bool operator<(ll x) const { return p < x; }
};
struct lineContainer : multiset <line, less<>> {
        static const ll inf = LLONG_MAX; // for doubles inf = 1/0, div(a, b) = a / b
        ll div(ll a, ll b) { // floored division
                return a / b - ((a ^ b) < 0 && a % b);
        }
        bool isect(iterator x, iterator y) {
                if (y == end()) return x->p = inf, 0;
                if (x->a == y->a) x->p = x->b > y->b ? inf : -inf;
                else x->p = div(y->b - x->b, x->a - y->a);
                return x->p >= y->p;
        }
        void add(ll a, ll b) { // do NOT use insert
                auto z = insert({a, b, 0}), y = z++, x = y;
                while (isect(y, z)) z = erase(z);
                if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
                while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
        }
        ll query(ll x) { // max value
                assert(!empty()); // or return -inf
                auto it = lower_bound(x);
                assert(it != end());
                return it->a * x + it->b;
        }
};
```

23. Treap

c368c4baa09fccd69372d199d49b6d0c

```
struct treap /* treap z zad odwr sort */ {
        struct node {
                node *l, *r;
                int val, min, rev, size, rank;
                node (int val = 0) : val(val), min(val), rev(false), l(NULL), r(NULL),
                ↪  size(1), rank(rand()) {}
        };
        node pool[1000005];
        int head = 0;
        node* mynew (int val){
                pool[head] = node(val);
                return pool + head ++;
        }
        node* root = NULL;
        int size(node* u) { return u == NULL ? 0 : u->size; }
        int min(node* u) { return u == NULL ? 1e9 : u->min; }
        void pull(node* u){
                u->size = 1 + size(u->l) + size(u->r);
                u->min = std::min(u->val, std::min(min(u->l), min(u->r)));
        }
        void push(node* u){
                if (u->rev){
                        swap(u->l, u->r);
                        if (u->l != NULL) u->l->rev ^= 1;
                        if (u->r != NULL) u->r->rev ^= 1;
                        u->rev = false;
                }
        }
        pair <node*, node*> split(node* u, int k){
                if (u == NULL) return {NULL, NULL};
                push(u);
                if (k <= size(u->l)){
                        auto p = split(u->l, k);
                        u->l = p.s;
                        pull(u);
                        return {p.f, u};
                } else {
                        auto p = split(u->r, k - size(u->l) - 1);
                        u->r = p.f;
                        pull(u);
                        return {u, p.s};
                }
        }
        node* merge(node* a, node* b){
                if (a == NULL) return b;
                if (b == NULL) return a;
                push(a); push(b);
                if (a->rank > b->rank){
                        a->r = merge(a->r, b);
                        pull(a);
                        return a;
                } else {
                        b->l = merge(a, b->l);
                        pull(b);
                        return b;
                }
        }
        void push_back(int val){
                root = merge(root, mynew(val));
        }
        void pop_front(){
                root = split(root, 1).s;
        }
        int find_min(){
                int res = 0;
                node* u = root;
                push(u);
                while (u->val != u->min){
                        push(u);
                        if (min(u->l) < min(u->r))
                                u = u->l;
                        else {
                                res += size(u->l) + 1;
                                u = u->r;
```

```
                }
            }
            push(u);
            res += size(u->l);
            return res;
        }
        void reverse_pref(int k){
            auto p = split(root, k);
            p.f->rev ^= 1;
            root = merge(p.f, p.s);
        }
};
```

24. RMQ

c5795c066f68626eeff6d3f4d7939ab2

```
// uzycie - RMQ < int, max > R(n, t);
template < typename TP, const TP& (*F)(const TP&, const TP&) >
struct RMQ {
        int n; vector < TP > t;
        vector < vector < TP > > res;
        RMQ(vector < TP > T) : n(sz(T)), t(T) {
            res.push_back(t);
            for(int p = 2; p <= n; p <<= 1) {
                vector < TP > tmp(n - p + 1);
                for(int j = 0; j <= n - p; j++)
                    tmp[j] = F(res.back()[j], res.back()[j + p / 2]);
                res.push_back(tmp);
            }
        }
        TP query(int a, int b) {
            int d = b - a + 1, lg = __lg(d);
            return F(res[lg][a], res[lg][b - (1 << lg) + 1]);
        }
};
```

25. Wavelet

b1df6bfeb814126836a6397c6686a611

```
//array values can be negative too, use appropriate minimum and maximum value
struct wavelet_tree {
  int lo, hi;
  wavelet_tree *l, *r;
  int *b, *c, bsz, csz; // c holds the prefix sum of elements

  wavelet_tree() {
    lo = 1;
    hi = 0;
    bsz = 0;
    csz = 0, l = NULL;
    r = NULL;
  }

  void init(int *from, int *to, int x, int y) {
    lo = x, hi = y;
    if(from >= to) return;
    int mid = (lo + hi) >> 1;
    auto f = [mid](int x) {
      return x <= mid;
    };
    b = (int*)malloc((to - from + 2) * sizeof(int));
    bsz = 0;
    b[bsz++] = 0;
    c = (int*)malloc((to - from + 2) * sizeof(int));
    csz = 0;
    c[csz++] = 0;
    for(auto it = from; it != to; it++) {
      b[bsz] = (b[bsz - 1] + f(*it));
      c[csz] = (c[csz - 1] + (*it));
      bsz++;
      csz++;
    }
    if(hi == lo) return;
    auto pivot = stable_partition(from, to, f);
    l = new wavelet_tree();
    l->init(from, pivot, lo, mid);
    r = new wavelet_tree();
    r->init(pivot, to, mid + 1, hi);
  }
  //kth smallest element in [l, r]
  //for array [1,2,1,3,5] 2nd smallest is 1 and 3rd smallest is 2
  int kth(int l, int r, int k) {
    if(l > r) return 0;
    if(lo == hi) return lo;
    int inLeft = b[r] - b[l - 1], lb = b[l - 1], rb = b[r];
    if(k <= inLeft) return this->l->kth(lb + 1, rb, k);
    return this->r->kth(l - lb, r - rb, k - inLeft);
  }
  //count of numbers in [l, r] Less than or equal to k
  int LTE(int l, int r, int k) {
    if(l > r || k < lo) return 0;
    if(hi <= k) return r - l + 1;
    int lb = b[l - 1], rb = b[r];
    return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r - rb, k);
  }
  //count of numbers in [l, r] equal to k
  int count(int l, int r, int k) {
    if(l > r || k < lo || k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int lb = b[l - 1], rb = b[r];
    int mid = (lo + hi) >> 1;
    if(k <= mid) return this->l->count(lb + 1, rb, k);
    return this->r->count(l - lb, r - rb, k);
  }
  //sum of numbers in [l ,r] less than or equal to k
  int sum(int l, int r, int k) {
    if(l > r or k < lo) return 0;
    if(hi <= k) return c[r] - c[l - 1];
    int lb = b[l - 1], rb = b[r];
    return this->l->sum(lb + 1, rb, k) + this->r->sum(l - lb, r - rb, k);
  }
  ~wavelet_tree() {
    delete l;
    delete r;
  }
};
```

26. Manacher

bec3c3aa22e11af07ca65aba34f8f78e

```
// @s[0..n-1]  - napis długości @n, @r[0..2n-2] - tablica promieni palindromów.
// s: a   b   a   a   b   a   a   c   a   a   b   b   b   b   a   a   c   a   c
// r: 0 0 1 0 0 3 0 0 2 0 0 1 0 0 3 0 0 1 0 0 0 1 1 6 1 1 0 0 0 1 0 0 1 0 1 0 0
void Manacher(const char* s, int n) {
  for (int i = 0, m = 0, k = 0, p = 0; i < 2 * n - 1; m = i++ - 1) {
    while (p < k and i / 2 + r[m] != k)
      r[i++] = min(r[m--], (k + 1 - p++) / 2);
    while (k + 1 < n and p > 0 and s[k + 1] == s[p - 1]) k++, p--;
    r[i] = (k + 1 - p++) / 2;
  }
}
```

27. Z Algorithm

9fe16bd4cdf892cb3dd4902537141da2

```
vector<int> Zalgorithm(string s) {
    int n = s.size(), L = 1, R = 1;
    vector <int> z(n); z[0] = n;
    for (int i = 1; i < n; ++i) {
        if (i + z[i - L] < R) { z[i] = z[i - L]; }
        else {
            L = i, R = max(R, i);
            while (R < n && s[R] == s[R - i]) R++;
            z[i] = R - i;
        }
    }
    return z;
}
```

28. Tablica sufiksowa (KMR)

a6d411bcccc6d53dcf89d972e4166801

```
int s[N]; /* s[i] > 0, n = len(s), A = sigma(s) */
int n, A, sa[N], lcp[N], cnt[N];
vector<int> x, y;
bool dif(int a, int b, int k) {
    return y[a] != y[b] || (a + k <= n ? y[a + k] : -1) != (b + k <= n ? y[b + k] : -1);
} /* 1-indexed */
void build() {
    x.clear(); x.resize(max(A, n) + 2); y = x; // +2 enough?
    int j = 0;
    rep(i, 1, n) cnt[x[i] = s[i]]++;
    rep(i, 1, A) cnt[i] += cnt[i - 1];
    per(i, 1, n) sa[cnt[x[i]]--] = i;
    for (int k = 1; k < n; k *= 2) {
        int p = 0;
        rep(i, n - k + 1, n) y[++p] = i;
        rep(i, 1, n) if (sa[i] > k) y[++p] = sa[i] - k;
        rep(i, 1, A) cnt[i] = 0;
        rep(i, 1, n) cnt[x[i]]++;
        rep(i, 1, A) cnt[i] += cnt[i - 1];
        per(i, 1, n) sa[cnt[x[y[i]]]--] = y[i];
        swap(x, y);
        A = x[sa[1]] = 1;
        rep(i, 2, n) x[sa[i]] = (A += dif(sa[i - 1], sa[i], k));
        if (n == A) break;
    }
    rep(i, 1, n) {
        if (x[i] == n) { lcp[x[i]] = 0; continue; }
        int nxt = sa[x[i] + 1];
        while (max(i, nxt) + j <= n && s[i + j] == s[nxt + j]) j++;
        lcp[x[i]] = j; j = max(j - 1, 0);
    }
}
```

29. Minimal Cyclic Shift

d7152ce63a24c08698efc1d6614b9428

```
string min_cyclic_string(string s) {
    s += s; int n = s.size(), i = 0, ans = 0;
    while (i < n / 2) {
        ans = i; int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            k = s[k] < s[j] ? i : k + 1; j++;
        }
        while (i <= k) i += j - k;
    }
    return s.substr(ans, n / 2);
}
```

30. All substring LCS

c5282ef0a4b5607d9f10834d57249cee

```
const int N = 2002;
int f[N][N], g[N][N];
void solve(string s, string t)
{
    int  n = s.size(), m = t.size();
    s = "#" + s;
    t = "#" + t;
    for (int i = 1; i <= m; ++i) f[0][i] = i;
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            if (s[i] == t[j]) {
                f[i][j] = g[i][j - 1];
                g[i][j] = f[i - 1][j];
            }
            else {
                f[i][j] = max(f[i - 1][j], g[i][j - 1]);
                g[i][j] = min(g[i][j - 1], f[i - 1][j]);
            }
        }
    }
    for (int i = 1; i <= m; ++i) {
        for (int j = i, ans = 0; j <= m; ++j) {
            if (i > f[n][j]) ++ans;
            //ans is lcs of s and t[i, j]
        }
    }
}
```

31. Eer Tree

4688ffa1a4d943a6d38d6262ebdd7fd3

```
const int N = 1e6 + 5; // CUSTOM
const int A = 26; // CUSTOM
int nxt[N][A], fail[N], last[N], len[N], cnt, par[N];
char s[N];
void prepare(int n){ // 0 <= i <= n + 1
    rep(i, 0, n + 1){
        rep(j, 0, A - 1) nxt[i][j] = 0;
    }
    s[0] = '#'; // CUSTOM
    last[0] = 1;
    cnt = 1;
    fail[0] = fail[1] = 1;
    len[1] = -1;
}
void add(int n){
    int c = s[n] - 'a'; // CUSTOM
    int v = last[n - 1];
    while(s[n - len[v] - 1] != s[n]) v = fail[v];
    if(!nxt[v][c]){
        int now = ++cnt, k = fail[v];
        len[now] = len[v] + 2;
        while(s[n - len[k] - 1] != s[n]) k = fail[k];
        fail[now] = nxt[k][c]; nxt[v][c] = now;
        par[now] = v;
    }
    last[n] = nxt[v][c];
}
```

32. Aho Corasick

3cb0555f3a1aa0ed2039cc397cfb04e9

```
const int N = 1 << 20;
int cnt, fail[N], go[N][26];
void add(string s) {
        int u = 0;
        for (auto ch : s) {
                int c = ch - 'a';
                if (!go[u][c]) go[u][c] = ++cnt;
                u = go[u][c];
        }
}
void build() {
        queue<int> q; q.push(0);
        while (!q.empty()) {
                int u = q.front(); q.pop();
                rep(c, 0, 25) {
                        int &v = go[u][c];
                        if (v == 0) v = go[fail[u]][c];
                        else {
                                fail[v] = u == 0 ? 0 : go[fail[u]][c];
                                q.push(v);
                        }
                }
        }
}
```

33. Suffix Automata

9d23628122d8f32efb9e4496ad25441d

```
// len -> largest string length of the corresponding endpos-equivalent class
// link -> longest suffix that is another endpos-equivalent class.
// firstpos -> 1 indexed end position of the first occurrence of the largest string of
↪  that node
// minlen(v) -> smallest string of node v = len(link(v)) + 1
// terminal nodes -> store the suffixes
struct SuffixAutomaton {
    struct node {
        int len, link, firstpos;
        map<char, int> nxt;
    };
    int sz, last;
    vector<node> t;
    vector<int> terminal;
    vector<long long> dp;
    vector<vector<int>> g;
    SuffixAutomaton() {}
    SuffixAutomaton(int n) {
        t.resize(2 * n); terminal.resize(2 * n, 0);
        dp.resize(2 * n, -1); sz = 1; last = 0;
        g.resize(2 * n);
        t[0].len = 0; t[0].link = -1; t[0].firstpos = 0;
    }
    void extend(char c) {
        int p = last;
        if (t[p].nxt.count(c)) {
            int q = t[p].nxt[c];
            if (t[q].len == t[p].len + 1) {
                last = q;
                return;
            }
            int clone = sz++;
            t[clone] = t[q];
            t[clone].len = t[p].len + 1;
            t[q].link = clone;
            last = clone;
            while (p != -1 && t[p].nxt[c] == q) {
                t[p].nxt[c] = clone;
                p = t[p].link;
            }
            return;
        }
        int cur = sz++;
        t[cur].len = t[last].len + 1;
        t[cur].firstpos = t[cur].len;
        p = last;
        while (p != -1 && !t[p].nxt.count(c)) {
            t[p].nxt[c] = cur;
            p = t[p].link;
        }
        if (p == -1) t[cur].link = 0;
        else {
            int q = t[p].nxt[c];
```

```
            if (t[p].len + 1 == t[q].len) t[cur].link = q;
            else {
                int clone = sz++;
                t[clone] = t[q];
                t[clone].len = t[p].len + 1;
                while (p != -1 && t[p].nxt[c] == q) {
                    t[p].nxt[c] = clone;
                    p = t[p].link;
                }
                t[q].link = t[cur].link = clone;
            }
        }
        last = cur;
    }
    void build_tree() {
        for (int i = 1; i < sz; i++) g[t[i].link].push_back(i);
    }
    void build(string &s) {
        for (auto x: s) {
            extend(x);
            terminal[last] = 1;
        }
        build_tree();
    }
    long long cnt(int i) { //number of times i-th node occurs in the string
        if (dp[i] != -1) return dp[i];
        long long ret = terminal[i];
        for (auto &x: g[i]) ret += cnt(x);
        return dp[i] = ret;
    }
};
```

34. Haszer

fe61560dc8f080c45a20c2f584ab88f7

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "using H = ull;" instead if you think test data is random.
#define FOR(i, a, b) for (int i = (a); i < (b); i++)
using ull = uint64_t;
struct H {
        ull x; H(ull _x=0) : x(_x) {}
        H operator+(H o) { return x + o.x + (x + o.x < x); }
        H operator-(H o) { return *this + ~o.x; }
        H operator*(H o) { auto m = (__uint128_t)x * o.x;
                return H((ull)m) + (ull)(m >> 64); }
        ull get() const { return x + !~x; }
        bool operator==(H o) const { return get() == o.get(); }
        bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (ll)1e11+3; // (order ~ 3e9; random also ok)
struct Hasher {
        vector<H> ha, pw;
        Hasher(string &str) : ha(SZ(str)+1), pw(ha) {
                pw[0] = 1;
                FOR(i,0,SZ(str)) {
                        ha[i+1] = ha[i] * C + str[i];
                        pw[i+1] = pw[i] * C;
                }
        }
        H hashInterval(int a, int b) { // hash [a, b]
                return ha[b + 1] - ha[a] * pw[b + 1 - a];
        }
};

H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

35. Fast Haszer

b4cfda0347bf535ccac0d9cad7a808cc

```
using ull = unsigned long long;
struct H {
        ull x; H(ull x=0) : x(x) {}
        H operator+(H o) { return x + o.x + (x + o.x < x); }
        H operator-(H o) { return *this + ~o.x; }
        H operator*(H o) { auto m = (__uint128_t)x * o.x;
                return H((ull)m) + (ull)(m >> 64); }
        ull get() const { return x + !~x; }
        bool operator==(H o) const { return get() == o.get(); }
        bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (ll)1e11+3; // (order ~ 3e9; random also ok)
```

36. Yarin sieve

b0d947d27230ee185dfee5f3dbea4828

```
#define MAXSIEVE 100000000
#define MAXSIEVEHALF (MAXSIEVE/2)
#define MAXSQRT 5000 // sqrt(MAXSIEVE)/2
char a[MAXSIEVE/16+2];
#define isprime(n) (a[(n)>>4]&(1<<(((n)>>1)&7))) // n is odd
void yarin_sieve() {
  memset(a,255,sizeof(a)); a[0]=0xFE;
  for(int i=1;i<MAXSQRT;i++) if (a[i>>3]&(1<<(i&7))) {
  for(int j=i+i+i+1;j<MAXSIEVEHALF;j+=i+i+1) a[j>>3]&=~(1<<(j&7)); }
}
```

37. Arithmetic Progressions Sums

4e1e30dd460407d3ca69674351ea4e1f

```
ll sumsq(ll n) {
  return n / 2 * ((n - 1) | 1);
} // \sum_{i = 0}^{n - 1}{(a + d * i) / m}, O(log m)
ll floor_sum(ll a, ll d, ll m, ll n) {
  ll res = d / m * sumsq(n) + a / m * n; d %= m; a %= m;
  if (!d) return res; ll to = (n * d + a) / m;
  return res + (n - 1) * to - floor_sum(m - 1 - a, m, d, to);
} // \sum_{i = 0}^{n - 1}{(a + d * i) % m}
ll mod_sum(ll a, ll d, ll m, ll n) {
  a = ((a % m) + m) % m; d = ((d % m) + m) % m;
  return n * a + d * sumsq(n) - m * floor_sum(a, d, m, n);
}
```

38. Partitions

b9fc1294ad577760b8e822e37ab9fa3b

```
vector<mint> solve(int n) {
  vector<mint> ans(n + 1);
  vector<pair<int, int>> gp; // (sign, generalized pentagonal numbers)
  gp.emplace_back(0, 0);
  for (int i = 1; gp.back().second <= n; i++) {
    gp.emplace_back(i % 2 ? 1 : -1, i * (3 * i - 1) / 2);
    gp.emplace_back(i % 2 ? 1 : -1, i * (3 * i + 1) / 2);
  }
  ans[0] = 1;
  for (int i = 1; i <= n; i++) {
    for (auto it : gp) {
      if (i >= it.second) ans[i] += ans[i - it.second] * it.first;
      else break;
    } /* remember that ans[i] can be negative here */
  }
  return ans;
}
```

39. Chinese Remainder Theorem

5101903203c6e2f4fd3b35236a2dfc78

```
typedef long long T;
pair<T, T> gcd_ext(T a, T b) {
    if(b == 0) return { 1, 0 };
    auto p = gcd_ext(b, a % b);
    return { p.second, p.first - a / b * p.second };
}
pair<T, T> CRT(vector<pair<T, T>> con) {
    T k = 0, m = 1;
    for(auto c: con) {
        T k1 = k, m1 = m, k2 = c.first, m2 = c.second;
        auto q = gcd_ext(m1, m2);
        T gcd = m1 * q.first + m2 * q.second;
        m = m1 / gcd * m2;
        if(k1 % gcd != k2 % gcd) return { -1, -1 };
        k = (k1 / gcd) * m2 * q.second + (k2 / gcd) * m1 * q.first + k1 % gcd;
        k %= m; if(k < 0) k += m;
    }
    return { k, m };
}
```

40. Linear Mod Minimum

2957373b31cd6275496bf1def3df3cba

```
template <typename T> /* min {ax + b mod m | 0 <= x < n } */
T go(T n, const T &m, T a, T b, bool is_min = true, T p = 1, T q = 1) {
    if (a == 0) return b;
    if (is_min) {
        if (b >= a) {
            T t = (m - b + a - 1) / a;
            T c = (t - 1) * p + q;
            if (n <= c) return b;
            n -= c; b += a * t - m;
        } b = a - 1 - b;
    } else {
        if (b < m - a) {
            T t = (m - b - 1) / a;
            T c = t * p;
            if (n <= c) return a * ((n - 1) / p) + b;
            n -= c; b += a * t;
        } b = m - 1 - b;
    }
    T d = m / a;
    T c = go(n, a, m % a, b, !is_min, (d - 1) * p + q, d * p + q);
    return is_min ? a - 1 - c : m - 1 - c;
}
```

41. Rabin Miller

d378c452de140d65723cf4aced8b7bdc

```
/* 2, 7, 61 are enough for n < 2^32, helpers in Rho-Pollard */
bool rabin(LL n){ /* [pr] is a table of primes */
        if(n == 2) return 1;
        if(n < 2 || !(n & 1)) return false;
        LL s = 0, r = n - 1;
        for(; !(r & 1); r >>= 1, ++s);
        for(int i = 0; pr[i] < n && pr[i] < maxv; ++i) {
                LL cur = fast(pr[i], r, n), nxt;
                for(int j = 0; j < s; ++j) {
                        nxt = mul(cur, cur, n);
                        if(nxt == 1 && cur != 1 && cur != n - 1) return false;
                        cur = nxt;
                }
                if(cur != 1) return false;
        }
        return true;
}
```

42. Rho Pollard

7dae74eb54659f2c47829ef8593ad832

```
const int maxv = 40;
const int maxp = 400'007;
inline LL mod(LL a, LL n){
        if(a >= n)        a -= n; return a;
}
inline LL add(LL a, LL b, LL n){
        a += b; mod(a, n); return a;
}
inline LL mul(LL x, LL y, LL p) {
        LL ret = x * y - (LL)((long double)x * y / p + 0.5) * p;
        return ret < 0 ? ret + p : ret;
}
LL fast(LL x, LL k, LL p){
        LL ret = 1%p;
        for(; k > 0; k >>= 1, x = mul(x, x, p))
                (k & 1) && (ret = mul(ret, x, p));
        return ret;
}
LL factor(LL n) { /* finds a divisor of n */
        static LL seq[maxp];
```

```
        while(true){
                LL x = rand()%n, y = x, c = rand()%n;
                LL *px = seq, *py = seq, tim = 0, prd = 1;
                while(true){
                        *py++ = y = add(mul(y, y, n), c, n);
                        *py++ = y = add(mul(y, y, n), c, n);
                        if((x = *px++) == y) break;
                        LL tmp = prd;
                        prd = mul(prd, abs(y - x), n);
                        if(!prd) return gcd(tmp, n);
                        if((++tim) == maxv){
                                if((prd = gcd(prd, n)) > 1 && prd < n) return prd;
                                tim = 0;
                        }
                }
                if(tim && (prd = gcd(prd, n)) > 1 && prd < n) return prd;
        }
}
```

43. FFT

4d6c99e53e9c2eab4ea8fffea35c5218

```
typedef double T;
const T PI = acos(-1.0);
struct C {
    T re, im;
    C () {}
    C (T r) : re(r), im(0) {}
    C (T r, T i) : re(r), im(i) {}
    C operator * (const C &c) const {
        return C(re * c.re - im * c.im, im * c.re + re * c.im);
    }
    C operator + (const C &c) const {
        return C(re + c.re, im + c.im);
    }
    C operator - (const C &c) const {
        return C(re - c.re, im - c.im);
    }
    void operator += (const C &c) {
        re += c.re, im += c.im;
    }
    C conj() const {
        return C(re, -im);
    }
};
typedef vector < C > VC;
typedef vector < LL > VLL;
inline void FFT(C *a, int n, int dir) {
    for(int i = 0, j = 0; i < n; i++) {
        if(i > j) swap(a[i], a[j]);
        for(int k = n >> 1; (j ^= k) < k; k >>= 1);
    }
    for(int p = 2; p <= n; p <<= 1) {
        C wn(cos(2.0 * dir * PI / p), sin(2.0 * dir * PI / p));
        for(int k = 0; k < n; k += p) {
            C w = 1;
            for(int j = 0; j < (p >> 1); j++) {
                C xx = a[k + j];
                C yy = w * a[k + j + (p >> 1)];
                a[k + j] = xx + yy;
                a[k + j + (p >> 1)] = xx - yy;
                w = w * wn;
            }
        }
    }
}
void multiply(VLL &a, VLL &b, VLL &res)  {
    int n = max(a.size(), b.size()), p = 2;
    while((p >> 1) < n) p <<= 1;
    C *fa = new C[p + 4];
    for(int i = 0; i < p; i++) fa[i] = 0;
    for(int i = 0; i < sz(a); i++) fa[i] += C(a[i], 0);
    for(int i = 0; i < sz(b); i++) fa[i] += C(0, b[i]);
    FFT(fa, p, 1);
    for(int i = 0; i <= p / 2; i++) {
        C bp = fa[i] + fa[p - i == p ? 0 : p - i].conj();
        C _q = fa[p - i == p ? 0 : p - i] - fa[i].conj();
        C q(_q.im, _q.re);
        fa[i] = (bp * q) * C(0.25);
        if(i > 0) fa[p - i] = fa[i].conj();
    }
    FFT(fa, p, -1);
    res.resize(sz(a) + sz(b) - 1);
    for(int i = 0; i < sz(res); i++) {
        res[i] = round(fa[i].re / p);
    }
    delete [] fa;
}
```

44. FFT

e7aa770b0e04b5fb8e2ee1dacac4a4cf

```
/* Prec. error max_ans/1e15 (2.5e18) for (long) doubles, so int rounding works
for doubles with answers 0.5e15, e.g. for sizes 2^20 and RANDOM ints in [0,45k],
assuming DBL_MANT_DIG=53 and LDBL_MANT_DIG=64. Consider normalizing and brute.*/
#define REP(i,n) for(int i = 0; i < int(n); ++i)
typedef double ld; // 'long double' is 2.2 times slower
struct C { ld re, im;
        C operator * (const C & he) const {
                return C{re * he.re - im * he.im,
                                re * he.im + im * he.re};
        }
        void operator += (const C & he) { re += he.re; im += he.im; }
};
void dft(vector<C> & a, bool rev) {
        const int n = a.size();
        for(int i = 1, k = 0; i < n; ++i) {
                for(int bit = n / 2; (k ^= bit) < bit; bit /= 2);;
                if(i < k) swap(a[i], a[k]);
        }
        for(int len = 1, who = 0; len < n; len *= 2, ++who) {
                static vector<C> t[30];
                vector<C> & om = t[who];
                if(om.empty()) {
                        om.resize(len);
                        const ld ang = 2 * acosl(0) / len;
```

```
                REP(i, len) om[i] = i%2 || !who ?
                                C{cos(i*ang), sin(i*ang)} : t[who-1][i/2];
        }
        for(int i = 0; i < n; i += 2 * len)
                REP(k, len) {
                        const C x = a[i+k], y = a[i+k+len]
                                * C{om[k].re, om[k].im * (rev ? -1 :
                                ↪ 1)};
                        a[i+k] += y;
                        a[i+k+len] = C{x.re - y.re, x.im - y.im};
                }
    }
    if(rev) REP(i, n) a[i].re /= n;
}
template<typename T>vector<T> multiply(const vector<T> & a, const vector<T> & b,
            bool split = true, bool normalize = false) {
    if(a.empty() || b.empty()) return {};
    T big = 0; if(normalize) { // [0,B] into [-B/2, B/2]
        assert(a.size() == b.size()); // equal size!!!
        for(T x : a) big = max(big, x);
        for(T x : b) big = max(big, x);
        big /= 2;
    }
    int n = a.size() + b.size();
    vector<T> ans(n - 1);
    /* if(min(a.size(),b.size()) < 190) { // BRUTE FORCE
        REP(i, a.size()) REP(j, b.size()) ans[i+j] += a[i]*b[j];
        return ans; } */
    while(n&(n-1)) ++n;
    auto foo = [&](const vector<C> & w, int i, int k) {
        int j = i ? n - i : 0, r = k ? -1 : 1;
        return C{w[i].re + w[j].re * r, w[i].im
                    - w[j].im * r} * (k ? C{0, -0.5} : C{0.5, 0});
    };
    if(!split) { // standard fast version
        vector<C> in(n), done(n);
        REP(i, a.size()) in[i].re = a[i] - big;
        REP(i, b.size()) in[i].im = b[i] - big;
        dft(in, false);
        REP(i, n) done[i] = foo(in, i, 0) * foo(in, i, 1);
        dft(done, true);
        REP(i, ans.size()) ans[i] = is_integral<T>::value ?
                    llround(done[i].re) : done[i].re;
    //REP(i,ans.size())err=max(err,abs(done[i].re-ans[i]));
    }
    else { // Split big INTEGERS into pairs a1*M+a2,
        const T M = 1<<15; // where M = sqrt(max_absvalue).
        vector<C> t[2]; // This version is 2.2-2.5 times slower.
        REP(x, 2) {
            t[x].resize(n);
            auto & in = x ? b : a; // below use (in[i]-big) if normalized
            REP(i, in.size()) t[x][i]=C{ld(in[i]%M), ld(in[i]/M)};
            dft(t[x], false);
        }
        T mul = 1;
        for(int s = 0; s < 3; ++s, mul = (mul*M)%mod) {
            vector<C> prod(n);
            REP(x, 2) REP(y, 2) if(x + y == s) REP(i, n)
                prod[i] += foo(t[0], i, x) * foo(t[1], i, y);
            dft(prod, true); // remember: llround(prod[i].re)%MOD*mul !!!
            REP(i, ans.size()) ans[i]=
            ↪ (ans[i]+llround(prod[i].re)%mod*mul)%mod;
        }
    }
    if(normalize) {
        T so_far = 0;
        REP(i, ans.size()) {
            if(i < (int) a.size()) so_far += a[i] + b[i];
            else so_far -= a[i-a.size()] + b[i-a.size()];
            ans[i] += big * so_far - big * big * min(i + 1, (int) ans.size()
            ↪ - i);
        }
    }
    return ans;
}
```

45. NTT

34d2fd257bd18d89e7411ee474963513

```
const int mod = 998244353, gen = 3;
void ntt(vector <int>& A){
    int n = A.size();
    static vector <int> roots = {1, 1};
    while (roots.size() < n){
        int s = roots.size();
        roots.resize(2 * s);
        int tmp[] = {1, pw(gen, (mod - 1) / (2 * s))};
        for (int i = s; i < 2 * s; i++){
            roots[i] = 1ll * roots[i / 2] * tmp[i % 2] % mod;
        }
    }
    for (int i = 1, j = 0; i < n; i++){
        j ^= n - (1 << __lg(n - j - 1));
        if (i < j) swap(A[i], A[j]);
    }
    for (int k = 1; k < n; k *= 2){
        for (int i = 0; i < n; i += 2 * k){
            for (int j = 0; j < k; j++){
                const int tmp = 1ll * roots[j + k] * A[i + j + k] % mod;
                A[i + j + k] = A[i + j] + mod - tmp;
                A[i + j] += tmp;
                if (A[i + j] >= mod) A[i + j] -= mod;
                if (A[i + j + k] >= mod) A[i + j + k] -= mod;
            }
        }
    }
}
vector <int> convolve(vector <int> A, vector <int> B){
    if (A.empty() || B.empty()) return {};
    int s = A.size() + B.size() - 1;
    int n = 1 << (__lg(s - 1) + 1);
    int inv = pw(n, mod - 2);
    A.resize(n), ntt(A);
    B.resize(n), ntt(B);
    vector <int> C(n);
    for (int i = 0; i < n; i++){
        C[-i & (n - 1)] = 1ll * A[i] * B[i] % mod * inv % mod;
```

```
        }
        ntt(C);
        C.resize(s);
        return C;
}
```

46. Polynomial division

ea39ca4b66d63e4fcd3faa4f25f03fd6

```
vector <int> polydiv(vector <int> a, vector <int> b) {
        int n, m, i, k, s;
        n = a.size(); m = b.size();
        if (m > n) return {0};
        reverse(a.begin(), a.end());
        reverse(b.begin(), b.end());
        s = moddiv(1, b[0], MOD);
        for (int& x : b) x = modmul(x, s, MOD);
        k = n - m + 1;
        vector <int> r, v, g = {1};
        for (int w = 1; (1 << (w - 1)) < k; w++){
                r = polymul(g, g); r.resize(1 << w);
                v = b; v.resize(1 << w);
                r = polymul(r, v);
                r.resize(1 << w); g.resize(1 << w);
                polyadd(g, g); polysub(g, r);
        }
        g = polymul(g, a); g.resize(k);
        reverse(g.begin(), g.end());
        for (int& x : g) x = modmul(x, s, MOD);
        return g;
}
```

47. General convolutions

f5d3d55d696d726c8de1c3610bdedb0e

```
const int mod = 998244353;
const int inv2 = (mod + 1) / 2;
struct tran {
        virtual int size() {}
        virtual void apply(int* A, bool inverse, int jump = 1) {}
};
struct xor_tran : tran {
        int size() { return 2; }
        void apply(int* A, bool inverse, int jump = 1)
        {
                int& na = A[0 * jump];
                int& nb = A[1 * jump];
                cint pa = na, pb = nb;
                na = pa + pb, nb = pa - pb;
                // na = pb, nb = pa + pb; and_tran
                // na = -pa + pb, nb = pa; inverse and_tran
                // na = pa + pb, nb = pa; or_tran
                // na = pb, nb = pa - pb; inverse or_tran
                if (na >= mod) na -= mod;
                if (nb < 0) nb += mod;
                if (inverse){
                        na = 1ll * na * inv2 % mod;
                        nb = 1ll * nb * inv2 % mod;
                }
        }
};
struct sum_tran : tran {
        int n;
        int size() { return n; }
        sum_tran(int s) : n(s) {}
        void apply(int* A, bool inverse, int jump = 1){
                vector <int> vec(n);
                for (int i = 0; i < n; i++) vec[i] = A[i * jump];
                ntt(vec);
                if (inverse){
                        int inv = pw(n, mod - 2);
                        reverse(vec.begin() + 1, vec.end());
                        for (int& a : vec) a = 1ll * a * inv % mod;
                }
                for (int i = 0; i < n; i++) A[i * jump] = vec[i];
        }
};
struct russian : tran // niezależne podkonwolucje (od najmłodszej do najstarszej)
{
        int n;
        vector <tran*> T;
        int size() { return n; }
        russian (vector <tran*> vec = {}){
                T = vec;
                n = 1;
                for (tran* t : T) n *= t->size();
        }
        void apply(int* A, bool inverse, int jump = 1){
                int s = 1;
                for (tran* t : T){
                        for (int i = 0; i < this->size(); i += s * t->size()){
                                for (int j = 0; j < s; j++){
                                        t->apply(A + (i + j) * jump, inverse, s * jump);
                                }
                        }
                        s *= t->size();
                }
        }
};
vector <int> convolve(tran* t, vector <int> A, vector <int> B){
        int n = t->size();
        assert(A.size() <= n && B.size() <= n);
        A.resize(n, 0), t->apply(&A[0], false);
        B.resize(n, 0), t->apply(&B[0], false);
        for (int i = 0; i < n; i++){
                A[i] = 1ll * A[i] * B[i] % mod;
        }
        t->apply(&A[0], true);
        return A;
}
```

48. Nim Multiplication

77bff48b5825c0c729cfd96df8d60c43

```
template <int L> inline u64 mulSlow(u64 a, u64 b) {
  static constexpr int l = L >> 1;
  const u64 a0 = a & ((1ULL << l) - 1), a1 = a >> l;
  const u64 b0 = b & ((1ULL << l) - 1), b1 = b >> l;
  const u64 a0b0 = mulSlow<l>(a0, b0);
  return (a0b0 ^ mulSlow<l>(1ULL << (l - 1), mulSlow<l>(a1, b1)))
      | (a0b0 ^ mulSlow<l>(a0 ^ a1, b0 ^ b1)) << l;
} /* O(log n^1.58), to optimize use (a ^ b) * c = a * c ^ b * c */
template <> inline u64 mulSlow<1>(u64 a, u64 b) { return a & b; }
```

49. Green Hackenbush

557ca3abdcd69b4d4c9b4b223dc96ce7

```
int n, T; /* Call init first, then set ground, then add edges */
vector <int> G[N];
int id[N], low[N], pre[N];
void set_ground(int u) { id[u] = 0; }
void add_edge(int u, int v) {
    G[id[u]].push_back(id[v]);
    if (id[u] != id[v]) G[id[v]].push_back(id[u]);
}
int dfs(int u, int p) {
    int ans = 0;
    pre[u] = low[u] = ++T;
    for (const int &v: G[u]) {
        if (v == p) { p = -1; }
        else if (pre[v] == 0) {
            int res = dfs(v, u);
            low[u] = min(low[u], low[v]);
            ans ^= low[v] > pre[u] ? res + 1 : (res ^ 1);
        } else if (pre[v] < pre[u]) {
            low[u] = min(low[u], pre[v]);
        } else { ans ^= 1; }
    }
    return ans;
}
void init(int _n) { n = _n; iota(id + 1, id + n + 1, 1); }
int run() { return dfs(0, -1); }
```

50. Red Blue Hackenbush

aa85012a35331797c1b1f69c95f26b77

```
struct Surreal {
    int value = 0, offset = 0;
    set <int> powers;
    void clear() { value = offset = 0; powers.clear(); }
    int size() { return powers.size(); }
    int sign() {
        const int tmp = 2 * value + !powers.empty();
        return tmp < 0 ? -1 : (tmp > 0);
    }
    int add_power(int power) {
        while (power) {
            if (!powers.count(power - offset)) {
                powers.insert(power - offset); break;
            }
            powers.erase(power - offset); --power;
        }
        return !power;
    }
    void operator += (const Surreal &v) {
        value += v.value;
        for (const int &power: v.powers) {
            value += add_power(power + v.offset);
        }
    }

    void divide(int power) {
        offset += power; int to_add = 0;
        for (int i = 0; i < power; ++i) {
            if (value & 1) { to_add += add_power(power - i); }
            value >>= 1;
        }
        value += to_add;
    }

    void get_next(int t) {
        int power = max(0, -t * value);
        value += t * (power + 1);
        if (value == -1 || (value == 1 && powers.empty())) {
            power++; value += t;
        }
        divide(power);
    }
};
struct RedBlueHack { /* Weights on edges should be -1 or 1 */
    int n;
    vector <int> id;
    vector <Surreal> ans;
    vector <vector <pair <int, int> > > G;
    RedBlueHack(int _n) : n(_n) {
        id.resize(n + 1); iota(id.begin(), id.end(), 0);
        ans.resize(n + 1); G.resize(n + 1);
    }
    void add_edge(int u, int v, int t) {
        G[u].push_back({v, t}); G[v].push_back({u, t});
    }
    void dfs(int u, int p) {
        ans[u].clear();
        for (auto &[v, w]: G[u]) {
            if (v == p) { continue; }
            dfs(v, u); ans[id[v]].get_next(w);
            if (ans[id[u]].size() < ans[id[v]].size()) {
                swap(id[u], id[v]);
            }
            ans[id[u]] += ans[id[v]];
        }
    }
    int run() { dfs(1, 1); return ans[id[1]].sign(); }
};
```

51. Berlekamp

0d0ab87e89132a8fab9e9a9b81638992

```
#define FOR(i, n) for(int i = 0; i < (n); i++)
#define sz(x) (int)(x).size()
const int N = 5005;
const int M = 1e9 + 7;
LL fast(LL a, LL n) {
    LL x = 1; a %= M;
    while(n) {
        if(n & 1) x = x * a % M;
        a = a * a % M; n >>= 1;
    }
    return x;
}
vector < LL > BM(vector < LL > x) {
    vector < LL > ls, cur;
    LL lf = 0, ld = 0;
    FOR(i, sz(x)) {
        LL t = 0;
        FOR(j, sz(cur)) t = (t + (LL)x[i - j - 1] * cur[j]) % M;
        if((t - x[i]) % M == 0) continue;
        if(cur.empty()) {
            cur.resize(i + 1);
            lf = i; ld = (t - x[i]) % M;
            continue;
        }
        LL k = -(x[i] - t) * fast(ld, M - 2) % M;
        vector < LL > c(i - lf - 1); c.push_back(k);
        for(auto y: ls) c.push_back(-y * k % M);
        if(sz(c) < sz(cur)) c.resize(sz(cur));
        FOR(j, sz(cur)) c[j] = (c[j] + cur[j]) % M;
        if(i - lf + sz(ls) >= sz(cur))
            ls = cur, lf = i, ld = (t - x[i]) % M;
        cur = c;
    }
    for(auto &y: cur) y = (y % M + M) % M;
    return cur;
}
int m;
LL a[N], h[N], t_[N], s[N], t[N];
void mull(LL *p, LL *q) {
    FOR(i, 2 * m) t_[i] = 0;
    FOR(i, m) if(p[i]) FOR(j, m)
        t_[i + j] = (t_[i + j] + p[i] * q[j]) % M;
    for(int i = 2 * m - 1; i >= m; i--) if(t_[i])
        for(int j = m - 1; ~j; --j)
            t_[i - j - 1] = (t_[i - j - 1] + t_[i] * h[j]) % M;
    FOR(i, m) p[i] = t_[i];
}
LL calc(LL k) {
    for(int i = m; ~i; i--) s[i] = t[i] = 0;
    s[0] = 1; (m != 1 ? t[1] = 1 : t[0] = h[0]);
    while(k) {
        if(k & 1) mull(s, t);
        mull(t, t); k >>= 1;
    }
    LL su = 0;
    FOR(i, m) su = (su + s[i] * a[i]) % M;
    return (su % M + M) % M;
}
LL nth_element(vector < LL > x, LL n) {
    if(n < (int)sz(x)) return x[n];
    vector < LL > v = BM(x); m = v.size(); if(!m) return 0;
    FOR(i, m) h[i] = v[i], a[i] = x[i];
    return calc(n);
}
```

52. Simplex

2c86cc90476d4687f17c0061f9956edf

```
const double EPS = 1e-7;
typedef long double T;
typedef vector <T> VT;
typedef vector < int > vi;
#define FOR(i,n) for(int i = 0; i < (n); i++)
vector < VT > A; VT b, c, res; vi kt, N; int m;
inline void pivot(int k, int l, int e) {
        int x = kt[l]; T p = A[l][e];
        FOR(i, k) A[l][i] /= p; b[l] /= p; N[e] = 0;
        FOR(i, m) if(i != l) b[i] -= A[i][e] * b[l], A[i][x] = A[i][e] * -A[l][x];
        FOR(j, k) if(N[j]) {
                c[j] -= c[e] * A[l][j];
                FOR(i, m) if(i != l) A[i][j] -= A[i][e] * A[l][j];
        }
        kt[l] = e; N[x] = 1; c[x] = c[e] * -A[l][x];
}
VT doit(int k) {
        VT res; T best;
        while(1) {
                int e = -1, l = -1; FOR(i, k) if(N[i] && c[i] > EPS) {e = i; break;}
                if(e == -1) break;
                FOR(i, m) if(A[i][e] > EPS && (l == -1 || best > b[i] / A[i][e]))
                        best = b[ l = i ]  / A[i][e];
                if(l == -1)
                        return VT();
                pivot(k, l, e);
        }
        res.resize(k, 0); FOR(i, m) res[kt[i]] = b[i];
        return res;
} /* AA * x <= bb, max cc * x */
VT simplex(const vector < VT > &AA, const VT &bb, const VT &cc) {
        int n = AA[0].size(), k;
        m = AA.size(); k = n + m + 1; kt.resize(m); b = bb; c = cc; c.resize(n + m);
        A = AA; FOR(i, m) { A[i].resize(k); A[i][n + i] = 1; A[i][k - 1] = -1; kt[i] = n
        ↪  + i; }
        N = vi(k, 1); FOR(i, m) N[kt[i]] = 0;
        int pos = min_element(b.begin(), b.end()) - b.begin();
        if(b[pos] < -EPS) {
                c = VT(k, 0); c[k - 1] = -1; pivot(k, pos, k - 1); res = doit(k);
                if(res[k - 1] > EPS) return VT();
                FOR(i, m) if (kt[i] == k - 1) {
                        FOR(j, k - 1) if(N[j] && (A[i][j] < -EPS || EPS < A[i][j])) {
                                pivot(k, i, j); break;
                        }
                }
                c = cc; c.resize(k, 0); FOR(i, m) FOR(j, k) if(N[j]) c[j] -= c[kt[i]] *
                ↪  A[i][j];
        }
        res = doit(k - 1); if(!res.empty()) res.resize(n);
```

```
        return res;
}
```

53. Josephus

```
    421dd32d75d61954519d94f3fc96fc4a
ll josephus(ll n, ll k, ll m) {
  m = n - m;
  if (k <= 1)return n - m;
  ll i = m;
  while (i < n) {
    ll r = (i - m + k - 2) / (k - 1);
    if ((i + r) > n) r = n - i;
    else if (!r) r = 1;
    i += r;
    m = (m + (r * k)) % i;
  } return m + 1;
}
```

54. K-th powers

```
    1654ffda4b65697bdf22c3861f39bec5
ll f(ll p, ll cnt, ll k) { /* distinct a^k % p^cnt over all a, prime p */
  if (cnt <= 0 or k == 0) return 1; /* O(cnt^2), optimize with precalc power */
  if (p == 2) {
    if (cnt == 1) return 2;
    ll u = power(2, cnt - 2) / __gcd(k, power(2, cnt - 2));
    if (k % 2) u *= 2;
    return u + f(2, cnt - k, k);
  }
  ll phi = power(p, cnt) - power(p, cnt - 1);
  ll u = phi / __gcd(k, phi);
  return u + f(p, cnt - k, k);
}
```

55. Multiplicative function sum

```
    ca2c8162f93b86c71e8efba7f759ce9a
/* p_f:the prefix sum of f(x) (1 <= x <= th).
p_g:the prefix sum of g(x) (0 <= x <= N).
p_c:the prefix sum of (f * g)(x) (0 <= x <= N). */
struct prefix_mul { /* th - threshold ~ N^2/3 */
        typedef ll (*func) (ll);
        func p_f, p_g, p_c; ll n, th, inv;
        unordered_map <ll,ll> mem;
        prefix_mul(func p_f, func p_g, func p_c) : p_f(p_f), p_g(p_g), p_c(p_c) {}
        ll calc(ll x) {
                if (x <= th) return p_f(x);
                auto d = mem.find(x);
                if (d != mem.end()) return d->second;
                ll ans = 0;
                for (ll i = 2, la; i <= x; i = la + 1){
                        la = x / (x / i);
                        ans = ans + (p_g(la) - p_g(i - 1)) * calc(x / i);
                }
                ans = (p_c(x) - ans) / inv;
                return mem[x] = ans;
        }
        ll solve(ll n, ll th) {
                if (n <= 0) return 0;
                prefix_mul::n = n; prefix_mul::th = th;
                inv = p_g(1); return calc(n);
        }
};
```

56. Tonelli Shanks

```
    590808cb69dfaf0f5a4e607e57e7238c
int get(int p){
        int t = 2;
        while(fast(t, (p - 1) / 2, p) == 1) ++t;
        return t;
}
int dsr(int v, int p){         //sqrt(p) mod p, p is prime, -1 no solution
        if(v == 0) return 0;
        if(p == 2) return 1;
        if(fast(v, (p - 1) / 2, p) == p - 1) return -1;
        int q = p - 1, s = 0;
        while(!(q & 1)) q /= 2, ++s;
        if(s == 1) return fast(v, (p + 1) / 4, p);
        int z = get(p), m = s, t = fast(v, q, p);
        int c = fast(z, q, p), r = fast(v, (q + 1) / 2, p);
        while(t != 1){
                int tt = t, i = 0;
                while(tt != 1) { tt = (1LL * tt * tt)%p, ++i; }
                int b = fast(c, fast(2, m - i - 1, p - 1), p);
                int b2 = (1LL * b * b)%p;
                r = (1LL * r * b)%p;
                t = (1LL * t * b2)%p;
                c = b2, m = i;
        }
        return r;
}
```

57. Fractions binary search

```
    27ab3ee7f7e442ea886e57cc08c8bff6
struct Frac { ll p, q; };
template<class F>
Frac fracBS(F f, ll N) {
        bool dir = 1, A = 1, B = 1;
        Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
        if (f(lo)) return lo;
        assert(f(hi));
        while (A || B) {
                ll adv = 0, step = 1; // move hi if dir, else lo
                for (int si = 0; step; (step *= 2) >>= si) {
                        adv += step;
                        Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
                        if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
                                adv -= step; si = 2;
                        }
                }
```

```
                hi.p += lo.p * adv; hi.q += lo.q * adv;
                dir = !dir;
                swap(lo, hi);
                A = B; B = !!adv;
        }
        return dir ? hi : lo;
}
```

58. Knight Moves

d96e198f0fc92fd25b63ecf6f899d272

```
// Minimum number of knight moves from (x,y) to
// (0,0) in non-negative infinite chessboard
ll knight_move(ll x, ll y) {
  ll cnt = max({(x + 1) / 2, (y + 1) / 2, (x + y + 2) / 3});
  while((cnt % 2) != (x + y) % 2) cnt++;
  if(x == 1 && !y) return 3;
  if(y == 1 && !x) return 3;
  if(x == y && x == 2) return 4;
  return cnt;
}
```

59. Gauss

87567ff9d3ecfeee7258bf305d169152

```
typedef double T;
const T eps = 1e-8;
int n; vector <T> ans; vector <vector <T> > in;
void init(int _n, vector <vector <T> > _in, vector <T> _ans) {
        n = _n; in = _in; ans = _ans;
}

bool solve(){        //zwraca czy uklad jest rozwiazywalny
        for(int i = 0; i < n; ++i){
                int id = i;
                for(int j = i + 1; j < n; ++j)
                        if(abs(in[j][i]) > abs(in[id][i])) id = j;
                if(abs(in[id][i]) < eps) return false;
                for(int j = 0; j < n; ++j) swap(in[i][j], in[id][j]);
                swap(ans[i], ans[id]);
                for(int j = i + 1; j < n; ++j){
                        if(abs(in[j][i]) < eps) continue;
                        T mult = in[j][i] / in[i][i];
                        for(int k = i; k < n; ++k) in[j][k] -= mult * in[i][k];
                        ans[j] -= mult * ans[i];
                }
        }
        for(int i = n - 1; i >= 0; --i){
                for(int j = n - 1; j > i; --j) ans[i] -= ans[j] * in[i][j];
                ans[i] /= in[i][i];
        }
        return true;
}
```

60. Geometry

b626f804f5a1a84d2b6ea52368f10dab

```
lf cross(point a, point b) { return a.x * b.y - a.y * b.x; }
lf dot(point a, point b) { return a.x * b.x + a.y * b.y; }
lf len2(point v) { return dot(v, v); }
lf len(point v) { return sqrt(len2(v)); }
struct paramline {
    point p, v;
    paramline() = default;
    paramline(point a, point b) : p(a), v(b - a) {}
};
point project(paramline l, point u){
    return l.p + dot((u - l.p), l.v) * l.v / len2(l.v);
}
lf dist(paramline l, point u){
    return abs(cross(l.v, u - l.p) / len(l.v));
}
struct line {
    lf A, B, C;
    line() = default;
    line(point a, point b){
        point d = b - a;
        A = -d.y;
                B = d.x;
        C = -(A * a.x + B * a.y);
    }
    line(paramline l){
        A = -l.v.y;
                B = l.v.x;
        C = -(A * l.p.x + B * l.p.y);
    }
};
point intersect(line a, line b){
    lf norm = a.B * b.A - a.A * b.B;
    return point{a.B * b.C - a.C * b.B, a.C * b.A - a.A * b.C} / -norm;
}
lf dist(line l, point v){
    return abs(l.A * v.x + l.B * v.y + l.C) / sqrt(l.A * l.A + l.B * l.B);
}
struct circle { point o; lf r; };
pair <point, point> intersect(circle a, circle b){
    point v = b.o - a.o;
        lf d = len(v);
    lf x = (d * d + a.r * a.r - b.r * b.r) / (2 * d);
    lf y = sqrt(a.r * a.r - x * x);
    v = v / d;
        point u{-v.y, v.x};
    return {a.o + x * v + y * u, a.o + x * v - y * u};
}
pair <point, point> intersect(circle c, paramline l){
    point pr = project(l, c.o);
    lf t = sqrt(c.r * c.r - len2(pr - c.o));
    point v = l.v / len(l.v);
    return {pr + t * v, pr - t * v};
}
pair <point, point> tangent(circle c, point v){
    point d = v - c.o;
    lf r = sqrt(len2(d) - c.r * c.r);
```

```
    return intersect(c, circle{v, r});
}
using segment = paramline;
lf len(segment s){ return len(s.v); }
bool on_segment(segment s, point p){
    p = p - s.p;
    const lf eps = 1e-9, d = dot(s.v, p);
    return -eps <= d && d <= dot(s.v, s.v) + eps && cross(s.v, p) < eps;
}
lf segment_dist(segment s, point v){
    point p = project(s, v);
    if (on_segment(s, p)) return len(v - p);
    return min(len(s.p - v), len(s.p + s.v - v));
}
bool segment_intersect(segment a, segment b){
    const lf eps = 1e-9; // change to negative to exclude endpoints
    return cross(b.p - a.p, a.v) * cross(b.p + b.v - a.p, a.v) < eps &&
                cross(a.p - b.p, b.v) * cross(a.p + a.v - b.p, b.v) < eps;
}
```

61. Halfplane intersection

ae4d415c820daf5ae2c11ba043e100cb

```
const lf eps = 1e-9, inf = 1e9;
struct halfplane {
        point p, pq; // Passing point and direction. Halfplane is to the left.
        lf ang;
        halfplane() {}
        halfplane(point a, point b) : p(a), pq(b - a), ang(atan2l(pq.y, pq.x)) {}
        bool out(point r) { return cross(pq, r - p) < -eps; }
};
point inter(halfplane s, halfplane t){
        return s.p + cross((t.p - s.p), t.pq) / cross(s.pq, t.pq) * s.pq;
}
vector <point> halfcoat(vector <halfplane>& H){
        point box[4] = {
                point(+inf, +inf),
                point(-inf, +inf),
                point(-inf, -inf),
                point(+inf, -inf)
        };
        for (int i = 0; i < 4; i++){
                H.push_back(halfplane(box[i], box[(i + 1) % 4]));
        }
        sort(H.begin(), H.end(), [] (halfplane a, halfplane b) { return a.ang < b.ang;
        ↪  });
        deque<halfplane> dq;
        int len = 0;
        for (int i = 0; i < H.size(); i++){
                while (len > 1 && H[i].out(inter(dq[len - 1], dq[len - 2]))){
                        dq.pop_back();
                        len--;
                }
                while (len > 1 && H[i].out(inter(dq[0], dq[1]))){
                        dq.pop_front();
                        len--;
                }
                if (len > 0 && abs(cross(H[i].pq, dq[len - 1].pq)) < eps){
                        if (dot(H[i].pq, dq[len - 1].pq) < 0.0){
                                return vector<point>();
                        }
                        if (H[i].out(dq[len - 1].p)){
                                dq.pop_back();
                                len--;
                        }
                        else continue;
                }
                dq.push_back(H[i]);
                len++;
        }
        while (len > 2 && dq[0].out(inter(dq[len - 1], dq[len - 2]))){
                dq.pop_back();
                len--;
        }
        while (len > 2 && dq[len - 1].out(inter(dq[0], dq[1]))){
                dq.pop_front();
                len--;
        }
        if (len < 3) return vector<point>();
        vector<point> res(len);
        for(int i = 0; i < len; i++){
                res[i] = inter(dq[i], dq[(i + 1) % len]);
        }
        return res;
}
```

62. Hull

825f3f1b67ad65545fddb9241452400e

```
point rot90(point a) { return point(-a.y, a.x); }
vector <point> convex_hull(vector <point> points, bool strict){
        sort(all(points));
        vector <point> hull;
        rep(phase, 0, 1){
                int start = ss(hull);
                for (point p : points){
                        while (hull.size() >= start + 2){
                                ll iw = IW(p, hull.back(), hull[ss(hull) - 2]);
                                if (iw < 0 || iw == 0 && strict == false) break;
                                hull.pop_back();
                        }
                        hull.pb(p);
                }
                hull.pop_back();
                reverse(all(points));
        }
        if (ss(hull) == 2 && hull[0] == hull[1]){
                hull.pop_back();
        }
        return hull;
}
struct cht {
        vector <point> hull, vecs;
        void insert(point p) // maintains lower hull. p.x should increase
        {
                while (!vecs.empty() && dot(vecs.back(), p - hull.back()) <= 0)
```

```
                {
                        hull.pop_back();
                        vecs.pop_back();
                }
                if (!hull.empty()){
                        vecs.pb(rot90(p - hull.back()));
                }
                hull.pb(p);
        }
        ll query(point p) // minimum dot product
        {
                auto it = lower_bound(all(vecs), p, [] (point a, point b) {
                        return cross(a, b) > 0;
                });
                return dot(p, hull[it - vecs.begin()]);
        }
};
```

63. Intersection Area of Circle and Polygon

22d843c3f9879b0eb8292701a4534812

```
using P = Point<double>;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
        auto tri = [&](P p, P q) {
                auto r2 = r * r / 2;
                P d = q - p;
                auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
                auto det = a * a - b;
                if (det <= 0) return arg(p, q) * r2;
                auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
                if (t < 0 || 1 <= s) return arg(p, q) * r2;
                P u = p + d * s, v = p + d * t;
                return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
        };
        auto sum = 0.0;
        FOR(i,0,SZ(ps))
                sum += tri(ps[i] - c, ps[(i + 1) % SZ(ps)] - c);
        return sum;
}
```

64. Minimum enclosing circle

1aa727609a185b98055b3bc98a0ec011

```
/* minimal enclosing circle */
pair<P, double> mec(vector<P> ps) {
        shuffle(all(ps), mt19937(1337));
        P o = ps[0];
        double r = 0, EPS = 1 + 1e-8;
        FOR(i,0,SZ(ps)) if ((o - ps[i]).dist() > r * EPS) {
                o = ps[i], r = 0;
                FOR(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
                        o = (ps[i] + ps[j]) / 2;
                        r = (o - ps[i]).dist();
                        FOR(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
                                o = ccCenter(ps[i], ps[j], ps[k]);
                                r = (o - ps[i]).dist();
                        }
                }
        }
        return {o, r};
}
```

65. Manhattan MST

3dd86e0eb3b50c3a61b6e63bd1ce75cf

```
/* returns O(n) edges which contains MST in O(nlogn) */
using P = Point<int>;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
        vi id(SZ(ps));
        iota(all(id), 0);
        vector<array<int, 3>> edges;
        FOR(k,0,4) {
                sort(all(id), [&](int i, int j) {
                     return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
                map<int, int> sweep;
                for (int i : id) {
                        for (auto it = sweep.lower_bound(-ps[i].y);
                                        it != sweep.end(); sweep.erase(it++)) {
                                int j = it->second;
                                P d = ps[i] - ps[j];
                                if (d.y > d.x) break;
                                edges.pb({d.y + d.x, i, j});
                        }
                        sweep[-ps[i].y] = i;
                }
                for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
        }
        return edges;
}
```

66. Many segments intersection

fe47d29707f3168860ff38760ed19dea

```
/* Description: Finds one of the segments intersections. */
template<class T>
pii allIntersect(vector<pair<Point<T>, Point<T>>> a) {
        using P = Point<T>;
        vector<tuple<P, int, int>> e;
        FOR(i, 0, SZ(a)) {
                if(a[i].nd < a[i].st) swap(a[i].st, a[i].nd);
                e.pb({a[i].st, 0, i}), e.pb({a[i].nd, 1, i});
        }
        sort(all(e));
        auto cmp = [](auto bb, auto cc) {
                auto [bs, be] = bb.st;
                auto [cs, ce] = cc.st;
                P sh(max(bs.x, cs.x), 0);
                auto bv = be - bs, cv = ce - cs;
                T l = bv.cross(bs - sh), r = cv.cross(cs - sh);
                // care! M^3
                return (sgn(cv.x) ? cv.x : 1) * (sgn(bv.x) ? l : bs.y) <
                                (sgn(bv.x) ? bv.x : 1) * (sgn(cv.x) ? r : cs.y);
        };
```

```
        auto inter = [](auto bb, auto cc) {
                return segInter(bb.st, bb.nd, cc.st, cc.nd);
        };
        set<pair<pair<P, P>, int>, decltype(cmp)> s(cmp);
        for(auto &[_, tp, id]: e) {
                auto akt = a[id];
                if(!tp) {
                        auto it = s.lower_bound({akt, id});
                        if(it != end(s) && SZ(inter(it->st, akt)))
                                return {it->nd, id};
                        if(it != begin(s) && SZ(inter((*--it).st, akt)))
                                return {it->nd, id};
                        s.insert({akt, id});
                }
                else {
                        auto it = s.erase(s.find({akt, id}));
                        if(it != begin(s) && it != end(s) &&
                                SZ(inter(it->st, prev(it)->st)))
                                return {it->nd, prev(it)->nd};
                }
        }
        return {-1, -1};
}
```

67. 3D Hull

cde46a9f0091517efbee9d13ef0db7fe

```
/**
 * Description: Computes all faces of the 3-dimension hull of a point set.
 *  *No four points must be coplanar*, or else random results will be returned.
 *  All faces will point outwards.
 * Time: O(n^2)
 */
#include "Point3D.h"
using P3 = Point3D<double>;
struct PR {
        void ins(int x) { (a == -1 ? a : b) = x; }
        void rem(int x) { (a == x ? a : b) = -1; }
        int cnt() { return (a != -1) + (b != -1); }
        int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
        assert(SZ(A) >= 4);
        vector<vector<PR>> E(SZ(A), vector<PR>(SZ(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
        vector<F> FS;
        auto mf = [&](int i, int j, int k, int l) {
                P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
                if (q.dot(A[l]) > q.dot(A[i])) q = q * -1;
                F f{q, i, j, k};
                E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
                FS.pb(f);
        };
        FOR(i,0,4) FOR(j,i+1,4) FOR(k,j+1,4)
                mf(i, j, k, 6 - i - j - k);
        FOR(i,4,SZ(A)) {
                FOR(j,0,SZ(FS)) {
                        F f = FS[j];
                        if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
                                E(a,b).rem(f.c);
                                E(a,c).rem(f.b);
                                E(b,c).rem(f.a);
                                swap(FS[j--], FS.back());
                                FS.pop_back();
                        }
                }
                int nw = SZ(FS);
                FOR(j,0,nw) {
                        F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
                        C(a, b, c); C(a, c, b); C(b, c, a);
                }
        }
        for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
                A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
        return FS;
};
```

68. Polygon tangent

9d2fa5d6a74c97eb687a6ddd34a50226

```
/* Description: Polygon tangents from a given point.
 *  The polygon must be ccw and have no collinear points.
 *  Returns a pair of indices of the given polygon.
 *  Should work for a point on border (for a point being polygon vertex returns previous
↪  and next one).
 * Time: O(\log n)
 */
#include "Point.h"
#define pdir(i) (ph ? p - poly[(i)%n] : poly[(i)%n] - p)
#define cmp(i,j) sgn(pdir(i).cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P>
array<int, 2> polygonTangents(vector<P>& poly, P p) {
        auto bs = [&](int ph) {
                int n = sz(poly), lo = 0, hi = n;
                if (extr(0)) return 0;
                while(lo + 1 < hi) {
                        int m = (lo + hi) / 2;
                        if (extr(m)) return m;
                        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
                        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi:lo) = m;
                }
                return lo;
        };
        array<int, 2> res = {bs(0), bs(1)};
        if(res[0] == res[1]) res[0] = (res[0] + 1) % sz(poly);
        if(poly[res[0]] == p) res[0] = (res[0] + 1) % sz(poly);
        return res;
}
```

69. Formulas

Liczby pierwsze: $10^9+123, 10^9+321, 999999929, 999999937, 10^{18}+9$.

Sumy: $\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$, $\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}$, $\sum_{i=n}^{m} \binom{i}{n} = \binom{m+1}{n+1}$, $\sum_{i=0}^{k} \binom{n}{i}\binom{m}{k-i} = \binom{n+m}{k}$.

Całki: $\int \frac{1}{ax+b}dx = \frac{1}{a}\ln|ax+b|$, $\int \tan x dx = -\ln|\cos x|$, $\int \frac{1}{x^2+a^2}dx = \frac{1}{a}\arctan\frac{x}{a}$, $\int \frac{1}{x^2-a^2}dx = \frac{1}{2a}\ln|\frac{x-a}{x+a}|$, $\int \frac{1}{\sqrt{a^2-x^2}}dx = \arcsin\frac{x}{a}$, $\int \frac{1}{\sqrt{x^2+q}}dx = \ln|x+\sqrt{x^2+q}|$, $\int a^x dx = \frac{a^x}{\ln a}$.

Liczby Catalana: $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$, $C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$, $C_{n+1} = C_n\frac{4n+2}{n+2}$, $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 74290, \ldots$

Suma dzielników: $\sigma(n) = \sigma(p_1^{\alpha_1}, \ldots, p_k^{\alpha_k}) = \prod_{i=1}^{k} \frac{p_i^{\alpha_i+1}-1}{p_i-1}$.

Funkcja Eulera: $\phi(p^k) = p^k - p^{k-1}, \phi(ab) = \phi(a)\phi(b)$ dla $a \perp b, \sum_{d|n}\phi(d) = n$.

Funkcja Mobiusa: 1 dla liczb bezkwadratowych z parzystą liczbą czynników, -1 – nieparzystą, 0 dla liczb nie bezkwadratowych. Inaczej $\mu(n)$ to suma pierwotnych pierwiastków z jedności stopnia n, $\sum_{d|n}\mu(d) = 0$ dla $n > 1$.

Związek między ϕ a μ: $\phi(n) = \sum_{d|n}\mu(d)\frac{n}{d}$.

Programowanie liniowe: Dla program prymalnego $\max c^T x$ z warunkami $Ax \le b$, $x \ge 0$, program dualny to $\min b^T y$ z warunkami $A^T y \ge c$, $y \ge 0$. Z silnego twierdzenia o dualności: $\max c^T x = \min b^T y$.

Problem znaczków pocztowych: Niech a, b względnie pierwsze. Jest dokładnie $\frac{1}{2}(a-1)(b-1)$ liczb, których nie da się zapisać w postaci $ax+by (x, y \le 0)$. Największa z nich to $(a-1)(b-1)-1$.

Lemat Burnside'a: Liczba orbit grupy G na zbiorze X: $|X/G| = \frac{1}{|G|}\sum_{g\in G}|X_g|$, gdzie $X_g = \{x \in X : g(x) = x\}$. ("Średnia liczba punktów stałych")

Metoda Simpsona: $\int_a^{b=a+2h} f(x)dx = \frac{b-a}{6}(f(a)+4f(a+h)+f(b)) + O(h^5 f^{(4)}(\xi))$.

Liczby Stirlinga pierwszego rodzaju: Opisują liczbę sposobów na rozmieszczenie n liczb w k cyklach, $\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$.

Liczby Stirlinga drugiego rodzaju: Opisują liczbę sposobów podziału zbioru n elementowego na k niepustych podzbiorów, $\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$, $\left\{{n \atop k}\right\} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$.

Liczby Bella: Liczba podziałów zbioru n elementowego, $\mathcal{B}_{n+1} = \sum_{k=0}^{n}\binom{n}{k}\mathcal{B}_k$.

Nieuporządkowania: Permutacje bez elementu stałego, $!n = \lfloor \frac{n!}{e} + \frac{1}{2} \rfloor$.

Liczby harmoniczne: $H_n = \sum_{k=1}^{n}\frac{1}{k}$, $\frac{1}{2n+1} < H_n - \ln n - \gamma < \frac{1}{2n}$, $\gamma = 0.57721\,56649\,01532\,86060\,65120\ldots$

Wzór Picka: $P = W + \frac{B}{2} - 1$ gdzie P – pole, W — wewnętrzne, B – brzegowe.

Trygonometria: $\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$, $\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$, tw. sinusów: $\frac{a}{\sin(\alpha)} = \frac{b}{\sin(\beta)} = \frac{c}{\sin(\gamma)} = 2R$, tw. cosinusów: $c^2 = a^2 + b^2 - 2ab\cos(\gamma)$, $R = \frac{abc}{4S}$, $r = \frac{2S}{a+b+c}$, $S = \sqrt{s(s-a)(s-b)(s-c)}$, gdzie S – pole trójkąta, $s = \frac{a+b+c}{2}$, r, R – promień okręgu wpisanego/opisanego.

Reguła Warnsdorffa obchodzenia skoczkiem szachownicy: W każdym kroku idź na pole, z którego można zrobić najmniejszą liczbę ruchów do nieodwiedzonych pól.

Optymalizacja Knutha: $dp[i][j] = \min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$, potrzebujemy $opt[i][j-1] \le opt[i][j] \le opt[i+1][j]$, gdzie $opt[i][j]$ daje najmniejsze optymalne k dla $dp[i][j]$, wystarcza też $C[a][c] + C[b][d] \le C[a][d] + C[b][c]$ i $C[b][c] \le C[a][d]$, dla wszytskich $a \le b \le c \le d$.

Pokrycie wierzchołkowe i zbiór niezależny: Niech M, C, I – maksymalne skojarzenie, minimalne pokrycie wierzchołkowe i maksymalny zbiór niezależny, wtedy $|M| \le |C| = N - |I|$, równość zachodzi dla grafów dwudzielnych. Dodatkowo $C^c = I$ (zawsze). Znajdowanie C, I (dla grafu dwudzielnego (A, B)): łączymy źródło z wierzchołkami z A, wierzchołki z B z ujściem (przepustowość taka jak wagi wierzchołków lub 1 dla nieważonego), krawędzie między A i B mają przepustowość ∞. Znajdujemy minimalne cięcie (S, T). Wtedy $C = (A \cap T) \cup (B \cap S)$ i $I = (A \cap S) \cup (B \cap T)$.

Macierz sąsiedztwa a liczba drzew spinających: Niech macierz $T = [t_{ij}]$, gdzie t_{ij} to liczba krawędzi z wierzchołka i do j dla $i \ne j$, $t_{ii} = -\deg(i)$. Liczba drzew spinających jest równa wyznacznikowi macierzy T po usunięciu k-tego wiersza i k-tej kolumny (k dowolne). Uwaga: dla niespójnych odpalać osobno dla każdej spójnej składowej.

Macierz a perfect matching: Tutte matrix: $A_{ij} = x_{ij}$ if $(i,j) \in E$ and $i < j$, $-x_{ij}$ if $i > j$, 0 if there is no edge.

Tw. Erdősa-Gallai: Ciąg $d_1, d_2, ..., d_n$ $(n-1 \ge d_1 \ge \cdots \ge d_n \ge 0)$ jest ciągiem stopni wierzchołków pewnego nieskierowanego grafu prostego $\iff$ $2|\sum d_i$ i $(\forall k \in \{1, \ldots, n-1\})\sum_{i=1}^{k} d_i \le k(k-1) + \sum_{i=k+1}^{n}\min(k, d_i)$.

Liczby Bernoulliego: $B_0 = 1$; $\sum_{k=0}^{m}\binom{m+1}{k}B_k = 0$; $1, \frac{-1}{2}, \frac{1}{6}, 0, \frac{-1}{30}, ...$; $\sum_{v=1}^{n} v^k = \frac{1}{k+1}\sum_{j=0}^{k}\binom{k+1}{j}B_j n^{k+1-j}$

Dni tygodnia: 01.01.1600 – sobota, 01.01.1900 – poniedziałek, 13.06.2042 – piątek, 01.04.2008 – wtorekm 31.12.1999 - piątek, 01.01.3000 – środa, 04.04.2019 – czwartek (dzień finałów).