Contents

1 Basic

1.1 vimrc

```
set nu rnu cin ts=4 sw=4 autoread hls
sy on
map<leader>b :w<bar>!g++ -std=c++17 '%' -DKEV -fsanitize=
     undefined -o /tmp/.run<CR>
map<leader>r :w<bar>!cat 01.in && echo "---" && /tmp/.run < 01.
     in<CR>
map<leader>i :!/tmp/.run<CR>
map<leader>c I//<Esc>
map<leader>y :%y+<CR>
map<leader>l :%d<bar>0r ~/t.cpp<CR>
```

1.2 Default code

```
#include <bits/stdc++.h>
using namespace std;
using i64 = long long;
using ll = long long;
#define SZ(v) (ll)((v).size())
#define pb emplace_back
#define AI(i) begin(i), end(i)
#define X first
#define Y second
template<class T> bool chmin(T &a, T b) { return b < a && (a =
     b, true); }
template<class T> bool chmax(T &a, T b) { return a < b && (a =
     b, true); }
// debug
#ifdef KEV
#define DE(args...) kout("[ " + string(#args) + " ] = ", args)
void kout() { cerr << endl; }
template<class T, class ...U> void kout(T a, U ...b) { cerr <<
     a << ' ', kout(b...); }
template<class T> void debug(T l, T r) { while (l != r) cerr <<
      *l << " \n"[next(l)==r], ++l; }
#else
#define DE(...) 0
#define debug(...) 0
#endif
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());
//
int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    return 0;
}
```

1.3 Fast Integer Input

```
char buf[1 << 16], *p1 = buf, *p2 = buf;
char get() {
    if (p1 == p2) {
        p1 = buf;
        p2 = p1 + fread(buf, 1, sizeof(buf), stdin);
    }
    if (p1 == p2)
        return -1;
    return *p1++;
}
```

```cpp
char readChar() {
    char c = get();
    while (isspace(c))
        c = get();
    return c;
}
int readInt() {
    int x = 0;
    char c = get();
    while (!isdigit(c))
        c = get();
    while (isdigit(c)) {
        x = 10 * x + c - '0';
        c = get();
    }
    return x;
}
```

1.4 Pragma optimization

```cpp
#pragma GCC optimize("Ofast", "no-stack-protector", "no-math-
    errno", "unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,sse4.2,popcnt,abm,
    mmx,avx,tune=native,arch=core-avx2,tune=core-avx2")
#pragma GCC ivdep
```

2 Flows, Matching

2.1 Flow

```cpp
template <typename F>
struct Flow {
    static constexpr F INF = numeric_limits<F>::max() / 2;
    struct Edge {
        int to;
        F cap;
        Edge(int to, F cap) : to(to), cap(cap) {}
    };
    int n;
    vector<Edge> e;
    vector<vector<int>> adj;
    vector<int> cur, h;
    Flow(int n) : n(n), adj(n) {}
    bool bfs(int s, int t) {
        h.assign(n, -1);
        queue<int> q;
        h[s] = 0;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i : adj[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) { return true; }
                    q.push(v);
                }
            }
        }
        return false;
    }
    F dfs(int u, int t, F f) {
        if (u == t) { return f; }
        F r = f;
        for (int &i = cur[u]; i < int(adj[u].size()); i++) {
            int j = adj[u][i];
            auto [v, c] = e[j];
            if (c > 0 && h[v] == h[u] + 1) {
                F a = dfs(v, t, min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) { return f; }
            }
        }
        return f - r;
    }
    // can be bidirectional
    void addEdge(int u, int v, F cf = INF, F cb = 0) {
        adj[u].push_back(e.size()), e.emplace_back(v, cf);
        adj[v].push_back(e.size()), e.emplace_back(u, cb);
    }
    F maxFlow(int s, int t) {
        F ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, INF);
        }
        return ans;
    }
    // do max flow first
    vector<int> minCut() {
        vector<int> res(n);
        for (int i = 0; i < n; i++) { res[i] = h[i] != -1; }
        return res;
    }
};
```

2.2 MCMF

```cpp
template <typename Flow, typename Cost>
struct MinCostMaxFlow {
    static constexpr Flow flowINF = numeric_limits<Flow>::max()
        / 2;
    static constexpr Cost costINF = numeric_limits<Cost>::max()
        / 2;
    struct Edge {
        int to;
        Flow cap;
        Cost cost;
        Edge(int to, Flow cap, Cost cost) : to(to), cap(cap),
            cost(cost) {}
    };
    int n;
    vector<Edge> e;
    vector<vector<int>> g;
    vector<Cost> h, dis;
    vector<int> pre;
    MinCostMaxFlow(int n) : n(n), g(n) {}
    bool spfa(int s, int t) {
        dis.assign(n, costINF);
        pre.assign(n, -1);
        vector<int> q{s}, inq(n);
        dis[s] = 0;
        inq[s] = 1;
        for (int i = 0; i < int(q.size()); i++) {
            int u = q[i];
            inq[u] = 0;
            for (int j : g[u]) {
                auto [v, cap, cost] = e[j];
                if (Cost nd = dis[u] + cost; cap > 0 && nd <
                    dis[v]) {
                    dis[v] = nd;
                    pre[v] = j;
                    if (!inq[v]) {
                        q.push_back(v);
                        inq[v] = 1;
                    }
                }
            }
        }
        return dis[t] != costINF;
    }
    bool dijkstra(int s, int t) {
        dis.assign(n, costINF);
        pre.assign(n, -1);
        priority_queue<pair<Cost, int>, vector<pair<Cost, int
            >>, greater<>> pq;
        dis[s] = 0;
        pq.emplace(0, s);
        while (!pq.empty()) {
            auto [d, u] = pq.top();
            pq.pop();
            if (dis[u] != d) continue;
            for (int i : g[u]) {
                auto [v, cap, cost] = e[i];
                if (Cost nd = d + h[u] - h[v] + cost; cap > 0
                    && dis[v] > nd) {
                    dis[v] = nd;
                    pre[v] = i;
                    pq.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != costINF;
    }
    void addEdge(int u, int v, Flow cap, Cost cost) {
        g[u].push_back(e.size());
        e.emplace_back(v, cap, cost);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -cost);
```

```
    }
    pair<Flow, Cost> maxFlow(int s, int t) {
        Flow flow = 0;
        Cost cost = 0;
        while (spfa(s, t)) {
            Flow aug = flowINF;
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                aug = min(aug, e[pre[i]].cap);
            }
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                e[pre[i]].cap -= aug;
                e[pre[i] ^ 1].cap += aug;
            }
            flow += aug;
            cost += aug * dis[t];
        }
        return make_pair(flow, cost);
    }
};
```

2.3 GomoryHu Tree

```
auto gomory(int n, vector<array<int, 3>> e) {
    Flow<int, int> mf(n);
    for (auto [u, v, c] : e) { mf.addEdge(u, v, c, c); }
    vector<array<int, 3>> res;
    vector<int> p(n);
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < int(e.size()); j++) { mf.e[j << 1].
            cap = mf.e[j << 1 | 1].cap = e[j][2]; }
        int f = mf.maxFlow(i, p[i]);
        auto cut = mf.minCut();
        for (int j = i + 1; j < n; j++) { if (cut[i] == cut[j]
            && p[i] == p[j]) { p[j] = i; }}
        res.push_back({f, i, p[i]});
    }
    return res;
}
```

2.4 Global Minimum Cut

```
// O(V ^ 3)
template <typename F>
struct GlobalMinCut {
    static constexpr int INF = numeric_limits<F>::max() / 2;
    int n;
    vector<int> vis, wei;
    vector<vector<int>> adj;
    GlobalMinCut(int n) : n(n), vis(n), wei(n), adj(n, vector<
        int>(n)) {}
    void addEdge(int u, int v, int w){
        adj[u][v] += w;
        adj[v][u] += w;
    }
    int solve() {
        int sz = n;
        int res = INF, x = -1, y = -1;
        auto search = [&]() {
            fill(vis.begin(), vis.begin() + sz, 0);
            fill(wei.begin(), wei.begin() + sz, 0);
            x = y = -1;
            int mx, cur;
            for (int i = 0; i < sz; i++) {
                mx = -1, cur = 0;
                for (int j = 0; j < sz; j++) {
                    if (wei[j] > mx) {
                        mx = wei[j], cur = j;
                    }
                }
                vis[cur] = 1, wei[cur] = -1;
                x = y;
                y = cur;
                for (int j = 0; j < sz; j++) {
                    if (!vis[j]) {
                        wei[j] += adj[cur][j];
                    }
                }
            }
            return mx;
        };
        while (sz > 1) {
            res = min(res, search());
            for (int i = 0; i < sz; i++) {
                adj[x][i] += adj[y][i];
                adj[i][x] = adj[x][i];
            }
            for (int i = 0; i < sz; i++) {
                adj[y][i] = adj[sz - 1][i];
                adj[i][y] = adj[i][sz - 1];
            }
            sz--;
        }
        return res;
    }
};
```

2.5 Bipartite Matching

```
struct BipartiteMatching {
    int n, m;
    vector<vector<int>> adj;
    vector<int> l, r, dis, cur;
    BipartiteMatching(int n, int m) : n(n), m(m), adj(n), l(n,
        -1), r(m, -1), dis(n), cur(n) {}
    // come on, you know how to write this
    void addEdge(int u, int v) { adj[u].push_back(v); }
    void bfs() {}
    bool dfs(int u) {}
    int maxMatching() {}
    auto minVertexCover() {
        vector<int> L, R;
        for (int u = 0; u < n; u++) {
            if (dis[u] == -1) {
                L.push_back(u);
            } else if (l[u] != -1) {
                R.push_back(l[u]);
            }
        }
        return pair(L, R);
    }
};
```

2.6 GeneralMatching

```
struct GeneralMatching {
    int n;
    vector<vector<int>> adj;
    vector<int> match;
    GeneralMatching(int n) : n(n), adj(n), match(n, -1) {}
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    int maxMatching() {
        vector<int> vis(n), link(n), f(n), dep(n);
        auto find = [&](int u) {
            while (f[u] != u) { u = f[u] = f[f[u]]; }
            return u;
        };
        auto lca = [&](int u, int v) {
            u = find(u);
            v = find(v);
            while (u != v) {
                if (dep[u] < dep[v]) { swap(u, v); }
                u = find(link[match[u]]);
            }
            return u;
        };
        queue<int> q;
        auto blossom = [&](int u, int v, int p) {
            while (find(u) != p) {
                link[u] = v;
                v = match[u];
                if (vis[v] == 0) {
                    vis[v] = 1;
                    q.push(v);
                }
                f[u] = f[v] = p;
                u = link[v];
            }
        };
        auto augment = [&](int u) {
            while (!q.empty()) { q.pop(); }
            iota(f.begin(), f.end(), 0);
            fill(vis.begin(), vis.end(), -1);
            q.push(u), vis[u] = 1, dep[u] = 0;
            while (!q.empty()){
                int u = q.front();
                q.pop();
                for (auto v : adj[u]) {
                    if (vis[v] == -1) {
                        vis[v] = 0;
                        link[v] = u;
                        dep[v] = dep[u] + 1;
```

```
                        if (match[v] == -1) {
                            for (int x = v, y = u, tmp; y !=
                                -1; x = tmp, y = x == -1 ? -1
                                : link[x]) {
                                tmp = match[y], match[x] = y,
                                    match[y] = x;
                            }
                            return true;
                        }
                        q.push(match[v]), vis[match[v]] = 1,
                            dep[match[v]] = dep[u] + 2;
                    } else if (vis[v] == 1 && find(v) != find(u
                        )) {
                        int p = lca(u, v);
                        blossom(u, v, p), blossom(v, u, p);
                    }
                }
            }
            return false;
        };
        int res = 0;
        for (int u = 0; u < n; ++u) { if (match[u] == -1) { res
            += augment(u); } }
        return res;
    }
};
```

2.7 Kuhn Munkres

```
// need perfect matching or not : w intialize with -INF / 0
template <typename Cost>
struct KM {
    static constexpr Cost INF = numeric_limits<Cost>::max() /
        2;
    int n;
    vector<Cost> hl, hr, slk;
    vector<int> l, r, pre, vl, vr;
    queue<int> q;
    vector<vector<Cost>> w;
    KM(int n) : n(n), hl(n), hr(n), slk(n), l(n, -1), r(n, -1),
        pre(n), vl(n), vr(n),
        w(n, vector<Cost>(n, -INF)) {}
    bool check(int x) {
        vl[x] = true;
        if (l[x] != -1) {
            q.push(l[x]);
            return vr[l[x]] = true;
        }
        while (x != -1) { swap(x, r[l[x] = pre[x]]); }
        return false;
    }
    void bfs(int s) {
        fill(slk.begin(), slk.end(), INF);
        fill(vl.begin(), vl.end(), false);
        fill(vr.begin(), vr.end(), false);
        q = {};
        q.push(s);
        vr[s] = true;
        while (true) {
            Cost d;
            while (!q.empty()) {
                int y = q.front();
                q.pop();
                for (int x = 0; x < n; ++x) {
                    if (!vl[x] && slk[x] >= (d = hl[x] + hr[y]
                        - w[x][y])) {
                        pre[x] = y;
                        if (d != 0) {
                            slk[x] = d;
                        } else if (!check(x)) {
                            return;
                        }
                    }
                }
            }
            d = INF;
            for (int x = 0; x < n; ++x) { if (!vl[x] && d > slk
                [x]) { d = slk[x]; }}
            for (int x = 0; x < n; ++x) {
                if (vl[x]) {
                    hl[x] += d;
                } else {
                    slk[x] -= d;
                }
                if (vr[x]) { hr[x] -= d; }
            }
            for (int x = 0; x < n; ++x) { if (!vl[x] && !slk[x]
                && !check(x)) { return; }}
        }
    }
    void addEdge(int u, int v, Cost x) { w[u][v] = max(w[u][v],
        x); }
    Cost solve() {
        for (int i = 0; i < n; ++i) { hl[i] = *max_element(w[i
            ].begin(), w[i].end()); }
        for (int i = 0; i < n; ++i) { bfs(i); }
        Cost res = 0;
        for (int i = 0; i < n; ++i) { res += w[i][l[i]]; }
        return res;
    }
};
```

2.8 Flow Models

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source S and sink T.
  2. For each edge (x, y, l, u), connect $x \to y$ with capacity $u - l$.
  3. For each vertex v, denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If $in(v) > 0$, connect $S \to v$ with capacity $in(v)$, otherwise, connect $v \to T$ with capacity $-in(v)$.
     – To maximize, connect $t \to s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T. If $f \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
     – To minimize, let f be the maximum flow from S to T. Connect $t \to s$ with capacity ∞ and let the flow from S to T be f'. If $f + f' \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, f' is the answer.
  5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v$, $v \in G$ with capacity K
  4. For each edge (u, v, w) in G, connect $u \to v$ and $v \to u$ with capacity w
  5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  6. T is a valid answer if the maximum flow $f < K|V|$
- 0/1 quadratic programming

$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x \bar{y} + \sum_{xyx'y'} c_{xyx'y'} (x\bar{y} + x'\bar{y'})$$

  can be minimized by the mincut of the following graph:

  1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y.
  2. Create edge (x, y) with capacity c_{xy}.
  3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3 Data Structure

3.1 <ext/pbds>

```
#include <bits/extc++.h>
#include <ext/rope>
using namespace __gnu_pbds;
using namespace __gnu_cxx;
#include <ext/pb_ds/assoc_container.hpp>
typedef tree<int, null_type, std::less<int>, rb_tree_tag,
    tree_order_statistics_node_update> tree_set;
typedef cc_hash_table<int, int> umap;
typedef priority_queue<int> heap;

int main() {
  // rb tree
  tree_set s;
  s.insert(71); s.insert(22);
  assert(*s.find_by_order(0) == 22); assert(*s.find_by_order(1)
      == 71);
  assert(s.order_of_key(22) == 0); assert(s.order_of_key(71) ==
      1);
  s.erase(22);
  assert(*s.find_by_order(0) == 71); assert(s.order_of_key(71)
      == 0);
  // mergable heap
  heap a, b; a.join(b);
  // persistant
  rope<char> r[2];
  r[1] = r[0];
  std::string st = "abc";
```

```
    r[1].insert(0, st.c_str());
    r[1].erase(1, 1);
    std::cout << r[1].substr(0, 2) << std::endl;
    return 0;
}
```

3.2 Li Chao Tree

```
// edu13F MLE with non-deleted pointers
// [) interval because of negative numbers
constexpr i64 INF64 = 4e18;
struct Line {
    i64 a = -INF64, b = -INF64;
    i64 operator()(i64 x) const {
        if (a == -INF64 && b == -INF64) {
            return -INF64;
        } else {
            return a * x + b;
        }
    }
};
constexpr int INF32 = 1e9;
struct LiChao {
    static constexpr int N = 5e6;
    array<Line, N> st;
    array<int, N> lc, rc;
    int n = 0;
    void clear() { n = 0; node(); }
    int node() {
        st[n] = {};
        lc[n] = rc[n] = -1;
        return n++;
    }
    void add(int id, int l, int r, Line line) {
        int m = (l + r) / 2;
        bool lcp = st[id](l) < line(l);
        bool mcp = st[id](m) < line(m);
        if (mcp) { swap(st[id], line); }
        if (r - l == 1) { return; }
        if (lcp != mcp) {
            if (lc[id] == -1) {
                lc[id] = node();
            }
            add(lc[id], l, m, line);
        } else {
            if (rc[id] == -1) {
                rc[id] = node();
            }
            add(rc[id], m, r, line);
        }
    }
    void add(Line line, int l = -INF32 - 1, int r = INF32 + 1)
     {
        add(0, l, r, line);
    }
    i64 query(int id, int l, int r, i64 x) {
        i64 res = st[id](x);
        if (r - l == 1) { return res; }
        int m = (l + r) / 2;
        if (x < m && lc[id] != -1) {
            res = max(res, query(lc[id], l, m, x));
        } else if (x >= m && rc[id] != -1) {
            res = max(res, query(rc[id], m, r, x));
        }
        return res;
    }
    i64 query(i64 x, int l = -INF32 - 1, int r = INF32 + 1) {
        return query(0, l, r, x);
    }
};
```

3.3 Treap

```
struct Treap {
    array<Treap*, 2> ch = {nullptr, nullptr};
    Treap *fa = nullptr;
    int x, P;
    int sz = 1;
    bool rev = false;

    Treap(int x = 0) : x(x), P(rng()) {}
    friend int size(Treap* t) {
        return t ? t->sz : 0;
    }
    void apply() {
        rev ^= 1;
    }
    void push() {
        if (rev) {
            swap(ch[0], ch[1]);
            for (auto k : ch) {
                if (k) {
                    k->apply();
                }
            }
            rev = false;
        }
    }
    void pull() {
        sz = 1;
        for (auto *k : ch) {
            if (k) {
                sz += k->sz;
                k->fa = this;
            }
        }
    }
};

Treap* merge(Treap *l, Treap *r) {
    if (!l) { return r; }
    if (!r) { return l; }
    if (l->P > r->P) {
        l->push();
        l->ch[1] = merge(l->ch[1], r);
        l->pull();
        return l;
    } else {
        r->push();
        r->ch[0] = merge(l, r->ch[0]);
        r->pull();
        return r;
    }
}

pair<Treap*, Treap*> splitSize(Treap *t, int left) {
    if (t) { t->fa = nullptr; }
    if (size(t) <= left) { return {t, nullptr}; }
    t->push();
    Treap* a;
    Treap* b;
    int sl = size(t->ch[0]) + 1;
    if (sl <= left) {
        a = t;
        tie(a->ch[1], b) = splitSize(t->ch[1], left - sl);
    } else {
        b = t;
        tie(a, b->ch[0]) = splitSize(t->ch[0], left);
    }
    t->pull();
    return {a, b};
}
```

3.4 Link-Cut Tree

```
struct Splay {
    array<Splay*, 2> ch = {nullptr, nullptr};
    Splay* fa = nullptr;
    int sz = 1;
    bool rev = false;
    Splay() {}
    void applyRev(bool x) {
        if (x) {
            swap(ch[0], ch[1]);
            rev ^= 1;
        }
    }
    void push() {
        for (auto k : ch) {
            if (k) {
                k->applyRev(rev);
            }
        }
        rev = false;
    }
    void pull() {
        sz = 1;
        for (auto k : ch) {
            if (k) {
            }
        }
    }
    int relation() { return this == fa->ch[1]; }
    bool isRoot() { return !fa || fa->ch[0] != this && fa->ch
        [1] != this; }
```

```cpp
    void rotate() {
        Splay *p = fa;
        bool x = !relation();
        p->ch[!x] = ch[x];
        if (ch[x]) { ch[x]->fa = p; }
        fa = p->fa;
        if (!p->isRoot()) { p->fa->ch[p->relation()] = this; }
        ch[x] = p;
        p->fa = this;
        p->pull();
    }
    void splay() {
        vector<Splay*> s;
        for (Splay *p = this; !p->isRoot(); p = p->fa) { s.push_back(p->fa); }
        while (!s.empty()) {
            s.back()->push();
            s.pop_back();
        }
        push();
        while (!isRoot()) {
            if (!fa->isRoot()) {
                if (relation() == fa->relation()) {
                    fa->rotate();
                } else {
                    rotate();
                }
            }
            rotate();
        }
        pull();
    }
    void access() {
        for (Splay *p = this, *q = nullptr; p; q = p, p = p->fa) {
            p->splay();
            p->ch[1] = q;
            p->pull();
        }
        splay();
    }
    void makeRoot() {
        access();
        applyRev(true);
    }
    Splay* findRoot() {
        access();
        Splay *p = this;
        while (p->ch[0]) { p = p->ch[0]; }
        p->splay();
        return p;
    }
    friend void split(Splay *x, Splay *y) {
        x->makeRoot();
        y->access();
    }
    // link if not connected
    friend void link(Splay *x, Splay *y) {
        x->makeRoot();
        if (y->findRoot() != x) {
            x->fa = y;
        }
    }
    // delete edge if doesn't exist
    friend void cut(Splay *x, Splay *y) {
        split(x, y);
        if (x->fa == y && !x->ch[1]) {
            x->fa = y->ch[0] = nullptr;
            x->pull();
        }
    }
    bool connected(Splay *x, Splay *y) {
        return x->findRoot() == y->findRoot();
    }
};
```

4 Graph

4.1 2-Edge-Connected Components

```cpp
struct EBCC {
    int n, cnt = 0, T = 0;
    vector<vector<int>> adj, comps;
    vector<int> stk, dfn, low, id;
    EBCC(int n) : n(n), adj(n), dfn(n, -1), low(n), id(n, -1) {}
    void addEdge(int u, int v) { adj[u].push_back(v), adj[v].push_back(u); }
    void build() { for (int i = 0; i < n; i++) { if (dfn[i] == -1) { dfs(i, -1); }}}
    void dfs(int u, int p) {
        dfn[u] = low[u] = T++;
        stk.push_back(u);
        for (auto v : adj[u]) {
            if (v == p) { continue; }
            if (dfn[v] == -1) {
                dfs(v, u);
                low[u] = min(low[u], low[v]);
            } else if (id[v] == -1) {
                low[u] = min(low[u], dfn[v]);
            }
        }
        if (dfn[u] == low[u]) {
            int x;
            comps.emplace_back();
            do {
                x = stk.back();
                comps.back().push_back(x);
                id[x] = cnt;
                stk.pop_back();
            } while (x != u);
            cnt++;
        }
    }
};
```

4.2 3-Edge-Connected Components

```cpp
// DSU
struct ETCC {
    int n, cnt = 0;
    vector<vector<int>> adj, comps;
    vector<int> in, out, low, up, nx, id;
    ETCC(int n) : n(n), adj(n), in(n, -1), out(in), low(n), up(n), nx(in), id(in) {}
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void build() {
        int T = 0;
        DSU d(n);
        auto merge = [&](int u, int v) {
            d.join(u, v);
            up[u] += up[v];
        };
        auto dfs = [&](auto dfs, int u, int p) -> void {
            in[u] = low[u] = T++;
            for (auto v : adj[u]) {
                if (v == u) { continue; }
                if (v == p) {
                    p = -1;
                    continue;
                }
                if (in[v] == -1) {
                    dfs(dfs, v, u);
                    if (nx[v] == -1 && up[v] <= 1) {
                        up[u] += up[v];
                        low[u] = min(low[u], low[v]);
                        continue;
                    }
                    if (up[v] == 0) { v = nx[v]; }
                    if (low[u] > low[v]) { low[u] = low[v], swap(nx[u], v); }
                    while (v != -1) { merge(u, v); v = nx[v]; }
                } else if (in[v] < in[u]) {
                    low[u] = min(low[u], in[v]);
                    up[u]++;
                } else {
                    for (int &x = nx[u]; x != -1 && in[x] <= in[v] && in[v] < out[x]; x = nx[x]) {
                        merge(u, x);
                    }
                    up[u]--;
                }
            }
            out[u] = T;
        };
        for (int i = 0; i < n; i++) { if (in[i] == -1) { dfs(dfs, i, -1); }}
```

```
        for (int i = 0; i < n; i++) { if (d.find(i) == i) { id[
            i] = cnt++; }}
        comps.resize(cnt);
        for (int i = 0; i < n; i++) { comps[id[d.find(i)]].
            push_back(i); }
    }
};
```

4.3 Heavy-Light Decomposition

```
struct HLD {
    int n, cur = 0;
    vector<int> sz, top, dep, par, tin, tout, seq;
    vector<vector<int>> adj;
    HLD(int n) : n(n), sz(n, 1), top(n), dep(n), par(n), tin(n)
        , tout(n), seq(n), adj(n) {}
    void addEdge(int u, int v) { adj[u].push_back(v), adj[v].
        push_back(u); }
    void build(int root = 0) {
        top[root] = root, dep[root] = 0, par[root] = -1;
        dfs1(root), dfs2(root);
    }
    void dfs1(int u) {
        if (auto it = find(adj[u].begin(), adj[u].end(), par[u
            ]); it != adj[u].end()) {
            adj[u].erase(it);
        }
        for (auto &v : adj[u]) {
            par[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            sz[u] += sz[v];
            if (sz[v] > sz[adj[u][0]]) { swap(v, adj[u][0]); }
        }
    }
    void dfs2(int u) {
        tin[u] = cur++;
        seq[tin[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        tout[u] = cur - 1;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = par[top[u]];
            } else {
                v = par[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }
    int dist(int u, int v) { return dep[u] + dep[v] - 2 * dep[
        lca(u, v)]; }
    int jump(int u, int k) {
        if (dep[u] < k) { return -1; }
        int d = dep[u] - k;
        while (dep[top[u]] > d) { u = par[top[u]]; }
        return seq[tin[u] - dep[u] + d];
    }
    // u is v's ancestor
    bool isAncestor(int u, int v) { return tin[u] <= tin[v] &&
        tin[v] <= tout[u]; }
    // root's parent is itself
    int rootedParent(int r, int u) {
        if (r == u) { return u; }
        if (isAncestor(r, u)) { return par[u]; }
        auto it = upper_bound(adj[u].begin(), adj[u].end(), r,
            [&](int x, int y) {
            return tin[x] < tin[y];
        }) - 1;
        return *it;
    }
    // rooted at u, v's subtree size
    int rootedSize(int r, int u) {
        if (r == u) { return n; }
        if (isAncestor(u, r)) { return sz[u]; }
        return n - sz[rootedParent(r, u)];
    }
    int rootedLca(int r, int a, int b) { return lca(a, b) ^ lca
        (a, r) ^ lca(b, r); }
};
```

4.4 Centroid Decomposition

```
vector<int> sz(n), vis(n);
auto dfs1 = [&](auto dfs1, int u, int p) -> void {
    sz[u] = 1;
    for (auto v : g[u]) {
        if (v != p && !vis[v]) {
            dfs1(dfs1, v, u);
            sz[u] += sz[v];
        }
    }
};
auto dfs2 = [&](auto dfs2, int u, int p, int tot) -> int {
    for (auto v : g[u]) {
        if (v != p && !vis[v] && 2 * sz[v] > tot) {
            return dfs2(dfs2, v, u, tot);
        }
    }
    return u;
};

auto dfs = [&](auto dfs, int cen) -> void {
    dfs1(dfs1, cen, -1);
    cen = dfs2(dfs2, cen, -1, sz[cen]);
    vis[cen] = 1;
    dfs1(dfs1, cen, -1);

    for (auto v : g[cen]) {
        if (!vis[v]) {
            dfs(dfs, v);
        }
    }
};
dfs(dfs, 0);
```

4.5 Strongly Connected Components

```
struct SCC {
    int n, cnt = 0, T = 0;
    vector<int> id, dfn, low, stk;
    vector<vector<int>> adj, comps;
    void addEdge(int u, int v) { adj[u].push_back(v); }
    SCC(int n) : n(n), id(n, -1), dfn(n, -1), low(n, -1), adj(n
        ) {}
    void build() {
        auto dfs = [&](auto dfs, int u) -> void {
            dfn[u] = low[u] = T++;
            stk.push_back(u);
            for (auto v : adj[u]) {
                if (dfn[v] == -1) {
                    dfs(dfs, v);
                    low[u] = min(low[u], low[v]);
                } else if (id[v] == -1) {
                    low[u] = min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u]) {
                int v;
                comps.emplace_back();
                do {
                    v = stk.back();
                    comps.back().push_back(v);
                    id[v] = cnt;
                    stk.pop_back();
                } while (u != v);
                cnt++;
            }
        };
        for (int i = 0; i < n; i++) { if (dfn[i] == -1) { dfs(
            dfs, i); }}
        reverse(comps.begin(), comps.end());
    }
    // the comps are in topological sorted order
};
```

4.6 2-SAT

```
struct TwoSat {
    int n, N;
    vector<vector<int>> adj;
    vector<int> ans;
    TwoSat(int n) : n(n), N(n), adj(2 * n) {}
    // u == x
    void addClause(int u, bool x) { adj[2 * u + !x].push_back(2
        * u + x); }
    // u == x || v == y
    void addClause(int u, bool x, int v, bool y) {
        adj[2 * u + !x].push_back(2 * v + y);
        adj[2 * v + !y].push_back(2 * u + x);
```

```
    }
    // u == x -> v == y
    void addImply(int u, bool x, int v, bool y) { addClause(u,
         !x, v, y); }
    void addVar() {
        adj.emplace_back(), adj.emplace_back();
        N++;
    }
    // at most one in var is true
    // adds prefix or as supplementary variables
    void atMostOne(const vector<pair<int, bool>> &vars) {
        int sz = vars.size();
        for (int i = 0; i < sz; i++) {
            addVar();
            auto [u, x] = vars[i];
            addImply(u, x, N - 1, true);
            if (i > 0) {
                addImply(N - 2, true, N - 1, true);
                addClause(u, !x, N - 2, false);
            }
        }
    }
    // does not return supplementary variables from atMostOne()
    bool satisfiable() {
        // run tarjan scc on 2 * N
        for (int i = 0; i < 2 * N; i++) { if (dfn[i] == -1) {
             dfs(dfs, i); }}
        for (int i = 0; i < N; i++) { if (id[2 * i] == id[2 * i
             + 1]) { return false; }}
        ans.resize(n);
        for (int i = 0; i < n; i++) { ans[i] = id[2 * i] > id[2
             * i + 1]; }
        return true;
    }
};
```

4.7 Minimum Mean Cycle

create a new vertex S, connect S to all vertices with arbitrary weight (0). Let $f_i(u)$ be the shortest path from S to u with exactly i edges.

$$ans = \min_{f_{n+1}(i)!=\infty} \max_{j=1}^{n} \frac{f_{n+1}(i) - f_j(i)}{n+1-j}$$

4.8 Minimum Steiner Tree

```
namespace steiner {
// Minimum Steiner Tree - O(N * 3^K + N^2 * 2^K)
// z[i] = the weight of the i-th vertex
const int maxn = 64, maxk = 10;
const int inf = 1e9;
int w[maxn][maxn], z[maxn], dp[1 << maxk][maxn], off[maxn];
void init(int n) {
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) w[i][j] = inf;
    z[i] = 0;
    w[i][i] = 0;
  }
}
void add_edge(int x, int y, int d) {
  w[x][y] = min(w[x][y], d);
  w[y][x] = min(w[y][x], d);
}
void build(int n) {
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      w[i][j] += z[i];
      if (i != j) w[i][j] += z[j];
    }
  }
  for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) w[i][j] = min(w[i][j], w[i][k
          ] + w[k][j] - z[k]);
    }
  }
}
int solve(int n, vector<int> mark) {
  build(n);
  int k = (int)mark.size();
  assert(k < maxk);
  for (int s = 0; s < (1 << k); ++s) {
    for (int i = 0; i < n; ++i) dp[s][i] = inf;
  }
  for (int i = 0; i < n; ++i) dp[0][i] = 0;
  for (int s = 1; s < (1 << k); ++s) {
    if (__builtin_popcount(s) == 1) {
      int x = __builtin_ctz(s);
      for (int i = 0; i < n; ++i) dp[s][i] = w[mark[x]][i];
      continue;
    }
    for (int i = 0; i < n; ++i) {
      for (int sub = s & (s - 1); sub; sub = s & (sub - 1)) {
        dp[s][i] = min(dp[s][i], dp[sub][i] + dp[s ^ sub][i] -
             z[i]);
      }
    }
    for (int i = 0; i < n; ++i) {
      off[i] = inf;
      for (int j = 0; j < n; ++j) off[i] = min(off[i], dp[s][j]
           + w[j][i] - z[j]);
    }
    for (int i = 0; i < n; ++i) dp[s][i] = min(dp[s][i], off[i
         ]);
  }
  int res = inf;
  for (int i = 0; i < n; ++i) res = min(res, dp[(1 << k) - 1][i
      ]);
  return res;
}}
```

4.9 Directed Minimum Spanning Tree

```
// DSU with rollback
template <typename Cost>
struct DMST {
    int n;
    vector<int> s, t, lc, rc, h;
    vector<Cost> c, tag;
    DMST(int n) : n(n), h(n, -1) {}
    void addEdge(int u, int v, Cost w) {
        int id = s.size();
        s.push_back(u), t.push_back(v), c.push_back(w);
        lc.push_back(-1), rc.push_back(-1);
        tag.emplace_back();
        h[v] = merge(h[v], id);
    }
    pair<Cost, vector<int>> build(int root = 0) {
        DSU d(n);
        Cost res{};
        vector<int> vis(n, -1), path(n), q(n), in(n, -1);
        vis[root] = root;
        vector<pair<int, vector<int>>> cycles;
        for (auto r = 0; r < n; ++r) {
            auto u = r, b = 0, w = -1;
            while (!~vis[u]) {
                if (!~h[u]) { return {-1, {}}; }
                push(h[u]);
                int e = h[u];
                res += c[e], tag[h[u]] -= c[e];
                h[u] = pop(h[u]);
                q[b] = e, path[b++] = u, vis[u] = r;
                u = d.find(s[e]);
                if (vis[u] == r) {
                    int cycle = -1, e = b;
                    do {
                        w = path[--b];
                        cycle = merge(cycle, h[w]);
                    } while (d.join(u, w));
                    u = d.find(u);
                    h[u] = cycle, vis[u] = -1;
                    cycles.emplace_back(u, vector<int>(q.begin
                        () + b, q.begin() + e));
                }
            }
            for (auto i = 0; i < b; ++i) { in[d.find(t[q[i]])]
                = q[i]; }
        }
        reverse(cycles.begin(), cycles.end());
        for (const auto &[u, comp] : cycles) {
            int count = int(comp.size()) - 1;
            d.back(count);
            int ine = in[u];
            for (auto e : comp) { in[d.find(t[e])] = e; }
            in[d.find(t[ine])] = ine;
        }
        vector<int> par;
        par.reserve(n);
        for (auto i : in) { par.push_back(i != -1 ? s[i] : -1);
          }
        return {res, par};
    }
    void push(int u) {
```

```
        c[u] += tag[u];
        if (int l = lc[u]; l != -1) { tag[l] += tag[u]; }
        if (int r = rc[u]; r != -1) { tag[r] += tag[u]; }
        tag[u] = 0;
    }
    int merge(int u, int v) {
        if (u == -1 || v == -1) { return u != -1 ? u : v; }
        push(u);
        push(v);
        if (c[u] > c[v]) { swap(u, v); }
        rc[u] = merge(v, rc[u]);
        swap(lc[u], rc[u]);
        return u;
    }
    int pop(int u) {
        push(u);
        return merge(lc[u], rc[u]);
    }
};
```

4.10 Maximum Clique

```
struct MaxClique {
  // change to bitset for n > 64.
  int n, deg[maxn];
  uint64_t adj[maxn], ans;
  vector<pair<int, int>> edge;
  void init(int n_) {
    n = n_;
    fill(adj, adj + n, 0ull);
    fill(deg, deg + n, 0);
    edge.clear();
  }
  void add_edge(int u, int v) {
    edge.emplace_back(u, v);
    ++deg[u], ++deg[v];
  }
  vector<int> operator()() {
    vector<int> ord(n);
    iota(ord.begin(), ord.end(), 0);
    sort(ord.begin(), ord.end(), [&](int u, int v) { return deg
        [u] < deg[v]; });
    vector<int> id(n);
    for (int i = 0; i < n; ++i) id[ord[i]] = i;
    for (auto e : edge) {
      int u = id[e.first], v = id[e.second];
      adj[u] |= (1ull << v);
      adj[v] |= (1ull << u);
    }
    uint64_t r = 0, p = (1ull << n) - 1;
    ans = 0;
    dfs(r, p);
    vector<int> res;
    for (int i = 0; i < n; ++i) {
      if (ans >> i & 1) res.push_back(ord[i]);
    }
    return res;
  }
#define pcount __builtin_popcountll
  void dfs(uint64_t r, uint64_t p) {
    if (p == 0) {
      if (pcount(r) > pcount(ans)) ans = r;
      return;
    }
    if (pcount(r | p) <= pcount(ans)) return;
    int x = __builtin_ctzll(p & -p);
    uint64_t c = p & ~adj[x];
    while (c > 0) {
      // bitset._Find_first(); bitset._Find_next();
      x = __builtin_ctzll(c & -c);
      r |= (1ull << x);
      dfs(r, p & adj[x]);
      r &= ~(1ull << x);
      p &= ~(1ull << x);
      c ^= (1ull << x);
    }
  }
};
```

4.11 Dominator Tree

```
vector<int> BuildDominatorTree(vector<vector<int>> g, int s) {
  int N = g.size();
  vector<vector<int>> rdom(N), r(N);
  vector<int> dfn(N, -1), rev(N, -1), fa(N, -1), sdom(N, -1),
       dom(N, -1), val(N, -1), rp(N, -1);
  int stamp = 0;
  auto Dfs = [&](auto dfs, int x) -> void {
    rev[dfn[x] = stamp] = x;
    fa[stamp] = sdom[stamp] = val[stamp] = stamp;
    stamp++;
    for (int u : g[x]) {
      if (dfn[u] == -1) {
        dfs(dfs, u);
        rp[dfn[u]] = dfn[x];
      }
      r[dfn[u]].push_back(dfn[x]);
    }
  };
  function<int(int, int)> Find = [&](int x, int c) {
    if (fa[x] == x) return c ? -1 : x;
    int p = Find(fa[x], 1);
    if (p == -1) return c ? fa[x] : val[x];
    if (sdom[val[x]] > sdom[val[fa[x]]]) val[x] = val[fa[x]];
    fa[x] = p;
    return c ? p : val[x];
  };
  auto Merge = [&](int x, int y) { fa[x] = y; };
  Dfs(Dfs, s);
  for (int i = stamp - 1; i >= 0; --i) {
    for (int u : r[i]) sdom[i] = min(sdom[i], sdom[Find(u, 0)])
      ;
    if (i) rdom[sdom[i]].push_back(i);
    for (int u : rdom[i]) {
      int p = Find(u, 0);
      if (sdom[p] == i) dom[u] = i;
      else dom[u] = p;
    }
    if (i) Merge(i, rp[i]);
  }
  vector<int> res(N, -2);
  res[s] = -1;
  for (int i = 1; i < stamp; ++i) {
    if (sdom[i] != dom[i]) dom[i] = dom[dom[i]];
  }
  for (int i = 1; i < stamp; ++i) res[rev[i]] = rev[dom[i]];
  return res;
}
```

4.12 Vizing's Theorem

```
// bipartite
e[i] = pair(u, v + a), deg[u]++, deg[v + a]++;
int col = *max_element(deg.begin(), deg.end());
vector<int> ans(m, -1);
vector has(a + b, vector<pair<int, int>>(col, {-1, -1}));
for (int i = 0; i < m; i++) {
    auto [u, v] = e[i];
    vector<int> c;
    for (auto x : {u, v}) {
        c.push_back(0);
        while (has[x][c.back()].first != -1) { c.back()++; }
    }
    if (c[0] != c[1]) {
        auto dfs = [&](auto dfs, int u, int x) -> void {
            auto [v, i] = has[u][c[x]];
            if (v != -1) {
                if (has[v][c[x ^ 1]].first != -1) {
                    dfs(dfs, v, x ^ 1);
                } else {
                    has[v][c[x]] = {-1, -1};
                }
                has[u][c[x ^ 1]] = {v, i}, has[v][c[x ^ 1]] = {
                    u, i};
                ans[i] = c[x ^ 1];
            }
        };
        dfs(dfs, v, 0);
    }
    has[u][c[0]] = {v, i};
    has[v][c[0]] = {u, i};
    ans[i] = c[0];
}
// general
auto vizing(int n, const vector<pair<int, int>> &e) {
    vector<int> deg(n);
    for (auto [u, v] : e) {
        deg[u]++, deg[v]++;
    }
    int col = *max_element(deg.begin(), deg.end()) + 1;
    vector<int> free(n);
    vector ans(n, vector<int>(n, -1));
    vector at(n, vector<int>(col, -1));
    auto update = [&](int u) {
```

```
        free[u] = 0;
        while (at[u][free[u]] != -1) {
            free[u]++;
        }
    };
    auto color = [&](int u, int v, int c1) {
        int c2 = ans[u][v];
        ans[u][v] = ans[v][u] = c1;
        at[u][c1] = v, at[v][c1] = u;
        if (c2 != -1) {
            at[u][c2] = at[v][c2] = -1;
            free[u] = free[v] = c2;
        } else {
            update(u), update(v);
        }
        return c2;
    };
    auto flip = [&](int u, int c1, int c2) {
        int v = at[u][c1];
        swap(at[u][c1], at[u][c2]);
        if (v != -1) {
            ans[u][v] = ans[v][u] = c2;
        }
        if (at[u][c1] == -1) {
            free[u] = c1;
        }
        if (at[u][c2] == -1) {
            free[u] = c2;
        }
        return v;
    };
    for (int i = 0; i < int(e.size()); i++) {
        auto [u, v1] = e[i];
        int v2 = v1, c1 = free[u], c2 = c1, d;
        vector<pair<int, int>> fan;
        vector<int> vis(col);
        while (ans[u][v1] == -1) {
            fan.emplace_back(v2, d = free[v2]);
            if (at[v2][c2] == -1) {
                for (int j = int(fan.size()) - 1; j >= 0; j--)
                    {
                    c2 = color(u, fan[j].first, c2);
                }
            } else if (at[u][d] == -1) {
                for (int j = int(fan.size()) - 1; j >= 0; j--)
                    {
                    color(u, fan[j].first, fan[j].second);
                }
            } else if (vis[d] == 1) {
                break;
            } else {
                vis[d] = 1, v2 = at[u][d];
            }
        }
        if (ans[u][v1] == -1) {
            while (v2 != -1) {
                v2= flip(v2, c2, d);
                swap(c2, d);
            }
            if (at[u][c1] != -1) {
                int j = int(fan.size()) - 2;
                while (j >= 0 && fan[j].second != c2) {
                    j--;
                }
                while (j >= 0) {
                    color(u, fan[j].first, fan[j].second);
                    j--;
                }
            } else {
                i--;
            }
        }
    }
    return pair(col, ans);
}
```

5 String

5.1 Prefix Function

```
template <typename T>
vector<int> prefixFunction(const T &s) {
    int n = int(s.size());
    vector<int> p(n);
    for (int i = 1; i < n; i++) {
        int j = p[i - 1];
        while (j > 0 && s[i] != s[j]) { j = p[j - 1]; }
        if (s[i] == s[j]) { j++; }
        p[i] = j;
    }
    return p;
}
```

5.2 Z Function

```
template <typename T>
vector<int> zFunction(const T &s) {
    int n = int(s.size());
    if (n == 0) return {};
    vector<int> z(n);
    for (int i = 1, j = 0; i < n; i++) {
        int& k = z[i];
        k = (j + z[j] <= i) ? 0 : min(j + z[j] - i, z[i - j]);
        while (i + k < n && s[k] == s[i + k]) { k++; }
        if (j + z[j] < i + z[i]) { j = i; }
    }
    z[0] = n;
    return z;
}
```

5.3 Suffix Array

```
struct SuffixArray {
    int n;
    vector<int> sa, as, ha;
    vector<vector<int>> rmq;
template <typename T>
    SuffixArray(const T &s) : n(s.size()), sa(n), as(n), ha(n -
        1) {
        n = s.size();
        iota(sa.begin(), sa.end(), 0);
        sort(sa.begin(), sa.end(), [&](int a, int b) { return s
            [a] < s[b]; });
        as[sa[0]] = 0;
        for (int i = 1; i < n; ++i) {
            as[sa[i]] = as[sa[i - 1]] + (s[sa[i]] != s[sa[i -
                1]]);
        }
        int k = 1;
        vector<int> tmp, cnt(n);
        tmp.reserve(n);
        while (as[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; ++i) { tmp.push_back(n - k +
                i); }
            for (auto i : sa) { if (i >= k) { tmp.push_back(i -
                k); } }
            fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; ++i) { ++cnt[as[i]]; }
            for (int i = 1; i < n; ++i) { cnt[i] += cnt[i - 1];
                }
            for (int i = n - 1; i >= 0; --i) { sa[--cnt[as[tmp[
                i]]]] = tmp[i]; }
            swap(as, tmp);
            as[sa[0]] = 0;
            for (int i = 1; i < n; ++i) {
                as[sa[i]] = as[sa[i - 1]] + (tmp[sa[i - 1]] <
                    tmp[sa[i]] || sa[i - 1] + k == n || tmp[sa
                    [i - 1] + k] < tmp[sa[i] + k]);
            }
            k *= 2;
        }
        for (int i = 0, j = 0; i < n; ++i) {
            if (as[i] == 0) {
                j = 0;
            } else {
                for (j -= j > 0; i + j < n && sa[as[i] - 1] + j
                    < n && s[i + j] == s[sa[as[i] - 1] + j];
                    ) { ++j; }
                ha[as[i] - 1] = j;
            }
        }
        if (n > 1) {
            const int lg = __lg(n - 1) + 1;
            rmq.assign(lg + 1, vector<int>(n - 1));
            rmq[0] = ha;
            for (int i = 1; i <= lg; i++) {
                for (int j = 0; j + (1 << i) < n; j++) {
                    rmq[i][j] = min(rmq[i - 1][j], rmq[i - 1][j
                        + (1 << i - 1)]);
                }
            }
```

```
        }
    }
    int lcp(int x, int y) {
        if (x == y) { return n - x; }
        x = as[x], y = as[y];
        if (x > y) { swap(x, y); }
        int k = __lg(y - x);
        return min(rmq[k][x], rmq[k][y - (1 << k)]);
    }
};
```

5.4 Manacher's Algorithm

```
int z[maxn];
int manacher(const string& s) {
  string t = ".";
  for (int i = 0; i < s.length(); ++i) t += s[i], t += '.';
  int l = 0, r = 0, ans = 0;
  for (int i = 1; i < t.length(); ++i) {
    z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
    while (i - z[i] >= 0 && i + z[i] < t.length() && t[i - z[i
        ]] == t[i + z[i]]) ++z[i];
    if (i + z[i] > r) r = i + z[i], l = i;
  }
  for (int i = 1; i < t.length(); ++i) ans = max(ans, z[i] - 1)
      ;
  return ans;
}
```

5.5 Aho-Corasick Automaton

```
constexpr int K = 26;
struct Node {
    array<int, K> nxt;
    int fail = -1;
    // other vars
    Node() { nxt.fill(-1); }
};
vector<Node> aho(1);
for (int i = 0; i < n; i++) {
    string s;
    cin >> s;
    int u = 0;
    for (auto ch : s) {
        int c = ch - 'a';
        if (aho[u].nxt[c] == -1) {
            aho[u].nxt[c] = aho.size();
            aho.emplace_back();
        }
        u = aho[u].nxt[c];
    }
}
vector<int> q;
for (auto &i : aho[0].nxt) {
    if (i == -1) {
        i = 0;
    } else {
        q.push_back(i);
        aho[i].fail = 0;
    }
}
for (int i = 0; i < int(q.size()); i++) {
    int u = q[i];
    if (u > 0) {
        // maintain
    }
    for (int c = 0; c < K; c++) {
        if (int v = aho[u].nxt[c]; v != -1) {
            aho[v].fail = aho[aho[u].fail].nxt[c];
            q.push_back(v);
        } else {
            aho[u].nxt[c] = aho[aho[u].fail].nxt[c];
        }
    }
}
```

5.6 Suffix Automaton

```
constexpr int K = 26;
struct Node{
    int len = 0, link = -1, cnt = 0;
    array<int, K> nxt;
    Node() { nxt.fill(-1); }
};
vector<Node> sam(1);
auto extend = [&](int c) {
    static int last = 0;
    int p = last, cur = sam.size();
    sam.emplace_back();
    sam[cur].len = sam[p].len + 1;
    sam[cur].cnt = 1;
    while (p != -1 && sam[p].nxt[c] == -1) {
        sam[p].nxt[c] = cur;
        p = sam[p].link;
    }
    if (p == -1) {
        sam[cur].link = 0;
    } else {
        int q = sam[p].nxt[c];
        if (sam[p].len + 1 == sam[q].len) {
            sam[cur].link = q;
        } else {
            int clone = sam.size();
            sam.emplace_back();
            sam[clone].len = sam[p].len + 1;
            sam[clone].link = sam[q].link;
            sam[clone].nxt = sam[q].nxt;
            while (p != -1 && sam[p].nxt[c] == q) {
                sam[p].nxt[c] = clone;
                p = sam[p].link;
            }
            sam[q].link = sam[cur].link = clone;
        }
    }
    last = cur;
};
for (auto ch : s) {
    extend(ch - 'a');
}
int N = sam.size();
vector<vector<int>> g(N);
for (int i = 1; i < N; i++) {
    g[sam[i].link].push_back(i);
}
```

5.7 Lexicographically Smallest Rotation

```
template <typename T>
T minRotation(T s) {
    int n = s.size();
    int i = 0, j = 1;
    s.insert(s.end(), s.begin(), s.end());
    while (i < n && j < n) {
        int k = 0;
        while (k < n && s[i + k] == s[j + k]) {
            k++;
        }
        if (s[i + k] <= s[j + k]) {
            j += k + 1;
        } else {
            i += k + 1;
        }
        if (i == j) {
            j++;
        }
    }
    int ans = i < n ? i : j;
    return T(s.begin() + ans, s.begin() + ans + n);
}
```

6 Math

6.1 Fast Fourier Transform

```
namespace fft {
struct cplx {
  double re, im;
  cplx(): re(0), im(0) {}
  cplx(double r, double i): re(r), im(i) {}
  cplx operator+(const cplx &rhs) const { return cplx(re + rhs.
      re, im + rhs.im); }
  cplx operator-(const cplx &rhs) const { return cplx(re - rhs.
      re, im - rhs.im); }
  cplx operator*(const cplx &rhs) const { return cplx(re * rhs.
      re - im * rhs.im, re * rhs.im + im * rhs.re); }
  cplx conj() const { return cplx(re, -im); }
};
const double pi = acos(-1);
cplx omega[maxn + 1];
void prefft() {
  for (int i = 0; i <= maxn; ++i)
```

```
    omega[i] = cplx(cos(2 * pi * i / maxn), sin(2 * pi * i /
        maxn));
}
void bitrev(vector<cplx> &v, int n) {
  int z = __builtin_ctz(n) - 1;
  for (int i = 0; i < n; ++i) {
    int x = 0;
    for (int j = 0; (1 << j) < n; ++j) x ^= (i >> j & 1) << (z
        - j);
    if (x > i) swap(v[x], v[i]);
  }
}
void fft(vector<cplx> &v, int n) {
  bitrev(v, n);
  for (int s = 2; s <= n; s <<= 1) {
    int z = s >> 1;
    for (int i = 0; i < n; i += s) {
      for (int k = 0; k < z; ++k) {
        cplx x = v[i + z + k] * omega[maxn / s * k];
        v[i + z + k] = v[i + k] - x;
        v[i + k] = v[i + k] + x;
      }
    }
  }
}
void ifft(vector<cplx> &v, int n) {
  fft(v, n);
  reverse(v.begin() + 1, v.end());
  for (int i = 0; i < n; ++i) v[i] = v[i] * cplx(1. / n, 0);
}
vector<long long> convolution(const vector<int> &a, const
    vector<int> &b) {
  // Should be able to handle N <= 10^5, C <= 10^4
  int sz = 1;
  while (sz < a.size() + b.size() - 1) sz <<= 1;
  vector<cplx> v(sz);
  for (int i = 0; i < sz; ++i) {
    double re = i < a.size() ? a[i] : 0;
    double im = i < b.size() ? b[i] : 0;
    v[i] = cplx(re, im);
  }
  fft(v, sz);
  for (int i = 0; i <= sz / 2; ++i) {
    int j = (sz - i) & (sz - 1);
    cplx x = (v[i] + v[j].conj()) * (v[i] - v[j].conj()) * cplx
        (0, -0.25);
    if (j != i) v[j] = (v[j] + v[i].conj()) * (v[j] - v[i].conj
        ()) * cplx(0, -0.25);
    v[i] = x;
  }
  ifft(v, sz);
  vector<long long> c(sz);
  for (int i = 0; i < sz; ++i) c[i] = round(v[i].re);
  return c;
}
vector<int> convolution_mod(const vector<int> &a, const vector<
    int> &b, int p) {
  int sz = 1;
  while (sz < (int)a.size() + (int)b.size() - 1) sz <<= 1;
  vector<cplx> fa(sz), fb(sz);
  for (int i = 0; i < (int)a.size(); ++i)
    fa[i] = cplx(a[i] & ((1 << 15) - 1), a[i] >> 15);
  for (int i = 0; i < (int)b.size(); ++i)
    fb[i] = cplx(b[i] & ((1 << 15) - 1), b[i] >> 15);
  fft(fa, sz), fft(fb, sz);
  double r = 0.25 / sz;
  cplx r2(0, -1), r3(r, 0), r4(0, -r), r5(0, 1);
  for (int i = 0; i <= (sz >> 1); ++i) {
    int j = (sz - i) & (sz - 1);
    cplx a1 = (fa[i] + fa[j].conj());
    cplx a2 = (fa[i] - fa[j].conj()) * r2;
    cplx b1 = (fb[i] + fb[j].conj()) * r3;
    cplx b2 = (fb[i] - fb[j].conj()) * r4;
    if (i != j) {
      cplx c1 = (fa[j] + fa[i].conj());
      cplx c2 = (fa[j] - fa[i].conj()) * r2;
      cplx d1 = (fb[j] + fb[i].conj()) * r3;
      cplx d2 = (fb[j] - fb[i].conj()) * r4;
      fa[i] = c1 * d1 + c2 * d2 * r5;
      fb[i] = c1 * d2 + c2 * d1;
    }
    fa[j] = a1 * b1 + a2 * b2 * r5;
    fb[j] = a1 * b2 + a2 * b1;
  }
  fft(fa, sz), fft(fb, sz);
  vector<int> res(sz);
  for (int i = 0; i < sz; ++i) {
    long long a = round(fa[i].re);
    long long b = round(fb[i].re);
    long long c = round(fa[i].im);
    res[i] = (a + ((b % p) << 15) + ((c % p) << 30)) % p;
  }
  return res;
}}
```

6.2 NTT and polynomials

```
template <int MOD>
struct Modint {
    static constexpr int P = MOD;
    int v;
    constexpr Modint() : v(0) {}
    constexpr Modint(i64 v_) : v(v_ % P) { if (v < 0) { v += P;
        }}
    constexpr friend Modint operator+(Modint a, Modint b) {
        return Modint((a.v + b.v) % P); }
    constexpr friend Modint operator-(Modint a, Modint b) {
        return Modint((a.v + P - b.v) % P); }
    constexpr friend Modint operator*(Modint a, Modint b) {
        return Modint(1LL * a.v * b.v % P); }
    constexpr Modint qpow(i64 p) {
        Modint res = 1, x = v;
        while (p > 0) {
            if (p & 1) { res = res * x; }
            x = x * x;
            p >>= 1;
        }
        return res;
    }
    constexpr Modint inv() { return qpow(P - 2); }
};
template<int P>
constexpr Modint<P> findPrimitiveRoot() {
    Modint<P> i = 2;
    int k = __builtin_ctz(P - 1);
    while (true) {
        if (i.qpow((P - 1) / 2).v != 1) { break; }
        i = i + 1;
    }
    return i.qpow(P - 1 >> k);
}
template<int P>
constexpr Modint<P> primitiveRoot = findPrimitiveRoot<P>();
vector<int> rev;
template<int P>
vector<Modint<P>> roots{0, 1};
template<int P>
void dft(vector<Modint<P>> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) { rev[i] = rev[i >> 1] >> 1
            | (i & 1) << k; }
    }
    for (int i = 0; i < n; i++) { if (rev[i] < i) { swap(a[i],
        a[rev[i]]); }}
    if (roots<P>.size() < n) {
        int k = __builtin_ctz(roots<P>.size());
        roots<P>.resize(n);
        while ((1 << k) < n) {
            auto e = Modint<P>(primitiveRoot<P>).qpow(P - 1 >>
                k + 1);
            for (int i = 1 << k - 1; i < 1 << k; i++) {
                roots<P>[2 * i] = roots<P>[i];
                roots<P>[2 * i + 1] = roots<P>[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                Modint<P> u = a[i + j];
                Modint<P> v = a[i + j + k] * roots<P>[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}
template <int P>
void idft(vector<Modint<P>> &a) {
```

```cpp
    int n = a.size();
    reverse(a.begin() + 1, a.end());
    dft(a);
    Modint<P> x = (1 - P) / n;
    for (int i = 0; i < n; i++) { a[i] = a[i] * x; }
}
template <int P>
struct Poly {
    using Mint = Modint<P>;
    vector<Mint> a;
    Poly() {}
    explicit Poly(const vector<Mint> &a) : a(a) {}
    explicit Poly(const initializer_list<Mint> &a) : a(a) {}
    explicit Poly(int n) : a(n) {}
template<class F>
    explicit Poly(int n, F f) : a(n) {
        for (int i = 0; i < n; i++) { a[i] = f(i); }
    }
    int size() const { return a.size(); }
    void resize(int n) { a.resize(n); }
    Mint operator[](int idx) const {
        if (idx < 0 || idx >= size()) { return 0; }
        return a[idx];
    }
    Mint& operator[](int idx) { return a[idx]; }
    Poly mulxk(int k) {
        auto b = a;
        b.insert(b.begin(), k, 0);
        return Poly(b);
    }
    Poly modxk(int k) {
        k = min(k, size());
        return Poly(vector<Mint>(a.begin(), a.begin() + k));
    }
    Poly divxk(int k) {
        if (size() <= k) { return Poly(); }
        return Poly(vector<Mint>(a.begin() + k, a.end()));
    }
    friend Poly operator+(const Poly &a, const Poly &b) {
        vector<Mint> res(max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); i++) { res[i] = a[
            i] + b[i]; }
        return Poly(res);
    }
    friend Poly operator-(const Poly &a, const Poly &b) {
        vector<Mint> res(max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); i++) { res[i] = a[
            i] - b[i]; }
        return Poly(res);
    }
    friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) { return Poly(); }
        int sz = 1, tot = a.size() + b.size() - 1;
        while (sz < tot) { sz *= 2; }
        a.resize(sz);
        b.resize(sz);
        dft(a.a);
        dft(b.a);
        for (int i = 0; i < sz; i++) { a[i] = a[i] * b[i]; }
        idft(a.a);
        a.resize(tot);
        return a;
    }
    friend Poly operator*(Mint a, Poly b) {
        for (int i = 0; i < int(b.size()); i++) { b[i] = b[i] *
            a; }
        return b;
    }
    friend Poly operator*(Poly a, Mint b) {
        for (int i = 0; i < int(a.size()); i++) { a[i] = a[i] *
            b; }
        return a;
    }
    Poly derivative()  {
        if (a.empty()) { return Poly(); }
        vector<Mint> res(size() - 1);
        for (int i = 0; i < size() - 1; ++i) { res[i] = (i + 1)
            * a[i + 1]; }
        return Poly(res);
    }
    Poly integral() {
        vector<Mint> res(size() + 1);
        for (int i = 0; i < size(); ++i) { res[i + 1] = a[i] *
            Mint(i + 1).inv(); }
        return Poly(res);
    }
    Poly inv(int m) {
        // a[0] != 0
        Poly x({a[0].inv()});
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly({2}) - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }
    Poly log(int m)  {
        return (derivative() * inv(m)).integral().modxk(m);
    }
    Poly exp(int m) {
        Poly x({1});
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly({1}) - x.log(k) + modxk(k))).modxk(k
                );
        }
        return x.modxk(m);
    }
    Poly pow(i64 k, int m) {
        if (k == 0) {
            vector<Mint> x(m);
            x[0] = 1;
            return Poly(x);
        }
        int i = 0;
        while (i < size() && a[i].v == 0) { i++; }
        if (i == size() || __int128(i) * k >= m) { return Poly(
            vector<Mint>(m)); }
        Mint v = a[i];
        auto f = divxk(i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i *
            k) * v.qpow(k);
    }
    Poly sqrt(int m) {
        // a[0] == 1, otherwise quadratic residue?
        Poly x({1});
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((Mint::
                P + 1) / 2);
        }
        return x.modxk(m);
    }
    Poly mulT(Poly b) const {
        if (b.size() == 0) { return Poly(); }
        int n = b.size();
        reverse(b.a.begin(), b.a.end());
        return ((*this) * b).divxk(n - 1);
    }
    vector<Mint> evaluation(vector<Mint> x) {
        if (size() == 0) { return vector<Mint>(x.size(), 0); }
        const int n = max(int(x.size()), size());
        vector<Poly> q(4 * n);
        vector<Mint> ans(x.size());
        x.resize(n);
        auto build = [&](auto build, int id, int l, int r) ->
            void {
            if (r - l == 1) {
                q[id] = Poly({1, -x[l].v});
            } else {
                int m = (l + r) / 2;
                build(build, 2 * id, l, m);
                build(build, 2 * id + 1, m, r);
                q[id] = q[2 * id] * q[2 * id + 1];
            }
        };
        build(build, 1, 0, n);
        auto work = [&](auto work, int id, int l, int r, const
            Poly &num) -> void {
            if (r - l == 1) {
                if (l < int(ans.size())) { ans[l] = num[0]; }
            } else {
                int m = (l + r) / 2;
                work(work, 2 * id, l, m, num.mulT(q[2 * id +
                    1]).modxk(m - l));
                work(work, 2 * id + 1, m, r, num.mulT(q[2 * id
                    ]).modxk(r - m));
            }
        };
        work(work, 1, 0, n, mulT(q[1].inv(n)));
```

```
        return ans;
    }
};
template <int P>
Poly<P> interpolate(vector<Modint<P>> x, vector<Modint<P>> y) {
    // f(xi) = yi
    int n = x.size();
    vector<Poly<P>> p(4 * n), q(4 * n);
    auto dfs1 = [&](auto dfs1, int id, int l, int r) -> void {
        if (l == r) {
            p[id] = Poly<P>({-x[l].v, 1});
            return;
        }
        int m = l + r >> 1;
        dfs1(dfs1, id << 1, l, m);
        dfs1(dfs1, id << 1 | 1, m + 1, r);
        p[id] = p[id << 1] * p[id << 1 | 1];
    };
    dfs1(dfs1, 1, 0, n - 1);
    Poly<P> f = Poly<P>(p[1].derivative().evaluation(x));
    auto dfs2 = [&](auto dfs2, int id, int l, int r) -> void {
        if (l == r) {
            q[id] = Poly<P>({y[l] * f[l].inv()});
            return;
        }
        int m = l + r >> 1;
        dfs2(dfs2, id << 1, l, m);
        dfs2(dfs2, id << 1 | 1, m + 1, r);
        q[id] = q[id << 1] * p[id << 1 | 1] + q[id << 1 | 1] *
            p[id << 1];
    };
    dfs2(dfs2, 1, 0, n - 1);
    return q[1];
}
```

6.3 NTT Prime List

Prime	Root	Prime	Root
7681	17	167772161	3
12289	11	104857601	3
40961	3	985661441	3
65537	3	998244353	3
786433	10	1107296257	10
5767169	3	2013265921	31
7340033	3	2810183681	11
23068673	3	2885681153	3
469762049	3	605028353	3

6.4 Newton's Method

Given $F(x)$ where

$$F(x) = \sum_{i=0}^{\infty} \alpha_i (x - \beta)^i$$

for β being some constant. Polynomial P such that $F(P) = 0$ can be found iteratively. Denote by Q_k the polynomial such that $F(Q_k) = 0 \pmod{x^{2^k}}$, then

$$Q_{k+1} = Q_k - \frac{F(Q_k)}{F'(Q_k)} \pmod{x^{2^{k+1}}}$$

6.5 Fast Walsh-Hadamard Transform

1. XOR Convolution
   - $f(A) = (f(A_0) + f(A_1), f(A_0) - f(A_1))$
   - $f^{-1}(A) = (f^{-1}(\frac{A_0+A_1}{2}), f^{-1}(\frac{A_0-A_1}{2}))$
2. OR Convolution
   - $f(A) = (f(A_0), f(A_0) + f(A_1))$
   - $f^{-1}(A) = (f^{-1}(A_0), f^{-1}(A_1) - f^{-1}(A_0))$
3. AND Convolution
   - $f(A) = (f(A_0) + f(A_1), f(A_1))$
   - $f^{-1}(A) = (f^{-1}(A_0) - f^{-1}(A_1), f^{-1}(A_1))$

6.6 Simplex Algorithm

Description: maximize $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} \le \mathbf{b}$ and $\mathbf{x} \ge 0$. Returns $-\infty$ if infeasible and ∞ if unbounded.

```
const double eps = 1e-9;
const double inf = 1e+9;
int n, m;
vector<vector<double>> d;
vector<int> p, q;
void pivot(int r, int s) {
  double inv = 1.0 / d[r][s];
  for (int i = 0; i < m + 2; ++i) {
    for (int j = 0; j < n + 2; ++j) {
      if (i != r && j != s) d[i][j] -= d[r][j] * d[i][s] * inv;
    }
  }
  for (int i = 0; i < m + 2; ++i) if (i != r) d[i][s] *= -inv;
  for (int j = 0; j < n + 2; ++j) if (j != s) d[r][j] *= +inv;
  d[r][s] = inv;
  swap(p[r], q[s]);
}
bool phase(int z) {
  int x = m + z;
  while (true) {
    int s = -1;
    for (int i = 0; i <= n; ++i) {
      if (!z && q[i] == -1) continue;
      if (s == -1 || d[x][i] < d[x][s]) s = i;
    }
    if (d[x][s] > -eps) return true;
    int r = -1;
    for (int i = 0; i < m; ++i) {
      if (d[i][s] < eps) continue;
      if (r == -1 || d[i][n + 1] / d[i][s] < d[r][n + 1] / d[r
          ][s]) r = i;
    }
    if (r == -1) return false;
    pivot(r, s);
  }
}
vector<double> solve(const vector<vector<double>> &a, const
     vector<double> &b, const vector<double> &c) {
  m = b.size(), n = c.size();
  d = vector<vector<double>>(m + 2, vector<double>(n + 2));
  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) d[i][j] = a[i][j];
  }
  p.resize(m), q.resize(n + 1);
  for (int i = 0; i < m; ++i) p[i] = n + i, d[i][n] = -1, d[i][
      n + 1] = b[i];
  for (int i = 0; i < n; ++i) q[i] = i, d[m][i] = -c[i];
  q[n] = -1, d[m + 1][n] = 1;
  int r = 0;
  for (int i = 1; i < m; ++i) if (d[i][n + 1] < d[r][n + 1]) r
      = i;
  if (d[r][n + 1] < -eps) {
    pivot(r, n);
    if (!phase(1) || d[m + 1][n + 1] < -eps) return vector<
        double>(n, -inf);
    for (int i = 0; i < m; ++i) if (p[i] == -1) {
      int s = min_element(d[i].begin(), d[i].end() - 1) - d[i].
          begin();
      pivot(i, s);
    }
  }
  if (!phase(0)) return vector<double>(n, inf);
  vector<double> x(n);
  for (int i = 0; i < m; ++i) if (p[i] < n) x[p[i]] = d[i][n +
      1];
  return x;
}
```

6.7 Subset Convolution

Description: $h(s) = \sum_{s' \subseteq s} f(s')g(s \setminus s')$

```
vector<int> SubsetConv(int n, const vector<int> &f, const
     vector<int> &g) {
  const int m = 1 << n;
  vector<vector<int>> a(n + 1, vector<int>(m)), b(n + 1, vector
      <int>(m));
  for (int i = 0; i < m; ++i) {
    a[__builtin_popcount(i)][i] = f[i];
    b[__builtin_popcount(i)][i] = g[i];
  }
  for (int i = 0; i <= n; ++i) {
    for (int j = 0; j < n; ++j) {
      for (int s = 0; s < m; ++s) {
        if (s >> j & 1) {
          a[i][s] += a[i][s ^ (1 << j)];
          b[i][s] += b[i][s ^ (1 << j)];
        }
      }
    }
  }
  vector<vector<int>> c(n + 1, vector<int>(m));
  for (int s = 0; s < m; ++s) {
    for (int i = 0; i <= n; ++i) {
```

```
      for (int j = 0; j <= i; ++j) c[i][s] += a[j][s] * b[i - j
          ][s];
    }
  }
  for (int i = 0; i <= n; ++i) {
    for (int j = 0; j < n; ++j) {
      for (int s = 0; s < m; ++s) {
        if (s >> j & 1) c[i][s] -= c[i][s ^ (1 << j)];
      }
    }
  }
  vector<int> res(m);
  for (int i = 0; i < m; ++i) res[i] = c[__builtin_popcount(i)
      ][i];
  return res;
}
```

6.7.1 Construction

Standard form: maximize $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} \le \mathbf{b}$ and $\mathbf{x} \ge 0$.
Dual LP: minimize $\mathbf{b}^T\mathbf{y}$ subject to $A^T\mathbf{y} \ge \mathbf{c}$ and $\mathbf{y} \ge 0$.
$\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ are optimal if and only if for all $i \in [1, n]$, either $\bar{x}_i = 0$ or $\sum_{j=1}^{m} A_{ji}\bar{y}_j = c_i$ holds and for all $i \in [1, m]$ either $\bar{y}_i = 0$ or $\sum_{j=1}^{n} A_{ij}\bar{x}_j = b_j$ holds.

1. In case of minimization, let $c'_i = -c_i$
2. $\sum_{1\le i\le n} A_{ji}x_i \ge b_j \to \sum_{1\le i\le n} -A_{ji}x_i \le -b_j$
3. $\sum_{1\le i\le n} A_{ji}x_i = b_j$
   - $\sum_{1\le i\le n} A_{ji}x_i \le b_j$
   - $\sum_{1\le i\le n} A_{ji}x_i \ge b_j$
4. If x_i has no lower bound, replace x_i with $x_i - x'_i$

6.8 Schreier–Sims Algorithm

```
namespace schreier {
int n;
vector<vector<vector<int>>> bkts, binv;
vector<vector<int>> lk;
vector<int> operator*(const vector<int> &a, const vector<int> &
    b) {
  vector<int> res(a.size());
  for (int i = 0; i < (int)a.size(); ++i) res[i] = b[a[i]];
  return res;
}
vector<int> inv(const vector<int> &a) {
  vector<int> res(a.size());
  for (int i = 0; i < (int)a.size(); ++i) res[a[i]] = i;
  return res;
}
int filter(const vector<int> &g, bool add = true) {
  n = (int)bkts.size();
  vector<int> p = g;
  for (int i = 0; i < n; ++i) {
    assert(p[i] >= 0 && p[i] < (int)lk[i].size());
    int res = lk[i][p[i]];
    if (res == -1) {
      if (add) {
        bkts[i].push_back(p);
        binv[i].push_back(inv(p));
        lk[i][p[i]] = (int)bkts[i].size() - 1;
      }
      return i;
    }
    p = p * binv[i][res];
  }
  return -1;
}
bool inside(const vector<int> &g) { return filter(g, false) ==
    -1; }
void solve(const vector<vector<int>> &gen, int _n) {
  n = _n;
  bkts.clear(), bkts.resize(n);
  binv.clear(), binv.resize(n);
  lk.clear(), lk.resize(n);
  vector<int> iden(n);
  iota(iden.begin(), iden.end(), 0);
  for (int i = 0; i < n; ++i) {
    lk[i].resize(n, -1);
    bkts[i].push_back(iden);
    binv[i].push_back(iden);
    lk[i][i] = 0;
  }
  for (int i = 0; i < (int)gen.size(); ++i) filter(gen[i]);
  queue<pair<pair<int, int>, pair<int, int>>> upd;
  for (int i = 0; i < n; ++i) {
    for (int j = i; j < n; ++j) {
      for (int k = 0; k < (int)bkts[i].size(); ++k) {
        for (int l = 0; l < (int)bkts[j].size(); ++l)
          upd.emplace(make_pair(i, k), make_pair(j, l));
      }
    }
  }
  while (!upd.empty()) {
    auto a = upd.front().first;
    auto b = upd.front().second;
    upd.pop();
    int res = filter(bkts[a.first][a.second] * bkts[b.first][b.
        second]);
    if (res == -1) continue;
    pair<int, int> pr = make_pair(res, (int)bkts[res].size() -
        1);
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < (int)bkts[i].size(); ++j) {
        if (i <= res) upd.emplace(make_pair(i, j), pr);
        if (res <= i) upd.emplace(pr, make_pair(i, j));
      }
    }
  }
}
long long size() {
  long long res = 1;
  for (int i = 0; i < n; ++i) res = res * bkts[i].size();
  return res;
}}
```

6.9 Berlekamp-Massey Algorithm

```
template <int P>
vector<int> BerlekampMassey(vector<int> x) {
  vector<int> cur, ls;
  int lf = 0, ld = 0;
  for (int i = 0; i < (int)x.size(); ++i) {
    int t = 0;
    for (int j = 0; j < (int)cur.size(); ++j)
      (t += 1LL * cur[j] * x[i - j - 1] % P) %= P;
    if (t == x[i]) continue;
    if (cur.empty()) {
      cur.resize(i + 1);
      lf = i, ld = (t + P - x[i]) % P;
      continue;
    }
    int k = 1LL * fpow(ld, P - 2, P) * (t + P - x[i]) % P;
    vector<int> c(i - lf - 1);
    c.push_back(k);
    for (int j = 0; j < (int)ls.size(); ++j)
      c.push_back(1LL * k * (P - ls[j]) % P);
    if (c.size() < cur.size()) c.resize(cur.size());
    for (int j = 0; j < (int)cur.size(); ++j)
      c[j] = (c[j] + cur[j]) % P;
    if (i - lf + (int)ls.size() >= (int)cur.size()) {
      ls = cur, lf = i;
      ld = (t + P - x[i]) % P;
    }
    cur = c;
  }
  return cur;
}
```

6.10 Fast Linear Recurrence

```
template <int P>
int LinearRec(const vector<int> &s, const vector<int> &coeff,
    int k) {
    int n = s.size();
    auto Combine = [&](const auto &a, const auto &b) {
        vector<int> res(n * 2 + 1);
        for (int i = 0; i <= n; ++i) {
            for (int j = 0; j <= n; ++j)
                (res[i + j] += 1LL * a[i] * b[j] % P) %= P;
        }
        for (int i = 2 * n; i > n; --i) {
            for (int j = 0; j < n; ++j)
                (res[i - 1 - j] += 1LL * res[i] * coeff[j] % P)
                    %= P;
        }
        res.resize(n + 1);
        return res;
    };
    vector<int> p(n + 1), e(n + 1);
    p[0] = e[1] = 1;
    for (; k > 0; k >>= 1) {
        if (k & 1) p = Combine(p, e);
        e = Combine(e, e);
```

```
    }
    int res = 0;
    for (int i = 0; i < n; ++i) (res += 1LL * p[i + 1] * s[i] % P) %= P;
    return res;
}
```

6.11 Prime check and factorize

```
i64 mul(i64 a, i64 b, i64 mod) {}
i64 qpow(i64 x, i64 p, i64 mod) {}
bool isPrime(i64 n) {
    if (n == 1) { return false; }
    int r = __builtin_ctzll(n - 1);
    i64 d = n - 1 >> r;
    auto checkComposite = [&](i64 p) {
        i64 x = qpow(p, d, n);
        if (x == 1 || x == n - 1) { return false; }
        for (int i = 1; i < r; i++) {
            x = mul(x, x, n);
            if (x == n - 1) { return false; }
        }
        return true;
    };
    for (auto p : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == p) {
            return true;
        } else if (checkComposite(p)) {
            return false;
        }
    }
    return true;
}
vector<i64> pollardRho(i64 n) {
    vector<i64> res;
    auto work = [&](auto work, i64 n) {
        if (n <= 10000) {
            for (int i = 2; i * i <= n; i++) {
                while (n % i == 0) {
                    res.push_back(i);
                    n /= i;
                }
            }
            if (n > 1) { res.push_back(n); }
            return;
        } else if (isPrime(n)) {
            res.push_back(n);
            return;
        }
        i64 x0 = 2;
        auto f = [&](i64 x) { return (mul(x, x, n) + 1) % n; };
        while (true) {
            i64 x = x0, y = x0, d = 1, power = 1, lam = 0, v = 1;
            while (d == 1) {
                y = f(y);
                ++lam;
                v = mul(v, abs(x - y), n);
                if (lam % 127 == 0) {
                    d = gcd(v, n);
                    v = 1;
                }
                if (power == lam) {
                    x = y;
                    power *= 2;
                    lam = 0;
                    d = gcd(v, n);
                    v = 1;
                }
            }
            if (d != n) {
                work(work, d);
                work(work, n / d);
                return;
            }
            ++x0;
        }
    };
    work(work, n);
    sort(res.begin(), res.end());
    return res;
}
```

6.12 Meissel–Lehmer Algorithm

```
int64_t PrimeCount(int64_t n) {
  if (n <= 1) return 0;
  const int v = sqrt(n);
  vector<int> smalls(v + 1);
  for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
  int s = (v + 1) / 2;
  vector<int> roughs(s);
  for (int i = 0; i < s; ++i) roughs[i] = 2 * i + 1;
  vector<int64_t> larges(s);
  for (int i = 0; i < s; ++i) larges[i] = (n / (2 * i + 1) + 1) / 2;
  vector<bool> skip(v + 1);
  int pc = 0;
  for (int p = 3; p <= v; ++p) {
    if (smalls[p] > smalls[p - 1]) {
      int q = p * p;
      pc++;
      if (1LL * q * q > n) break;
      skip[p] = true;
      for (int i = q; i <= v; i += 2 * p) skip[i] = true;
      int ns = 0;
      for (int k = 0; k < s; ++k) {
        int i = roughs[k];
        if (skip[i]) continue;
        int64_t d = 1LL * i * p;
        larges[ns] = larges[k] - (d <= v ? larges[smalls[d] - pc] : smalls[n / d]) + pc;
        roughs[ns++] = i;
      }
      s = ns;
      for (int j = v / p; j >= p; --j) {
        int c = smalls[j] - pc;
        for (int i = j * p, e = min(i + p, v + 1); i < e; ++i) smalls[i] -= c;
      }
    }
  }
  for (int k = 1; k < s; ++k) {
    const int64_t m = n / roughs[k];
    int64_t s = larges[k] - (pc + k - 1);
    for (int l = 1; l < k; ++l) {
      int p = roughs[l];
      if (1LL * p * p > m) break;
      s -= smalls[m / p] - (pc + l - 1);
    }
    larges[0] -= s;
  }
  return larges[0];
}
```

6.13 Discrete Logarithm

```
// return min x >= 0 s.t. a ^ x = b mod m, 0 ^ 0 = 1, -1 if no solution
// (I think) if you want x > 0 (m != 1), remove if (b == k) return add;
int discreteLog(int a, int b, int m) {
    if (m == 1) {
        return 0;
    }
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k) {
            return add;
        } else if (b % g) {
            return -1;
        }
        b /= g, m /= g, ++add;
        k = 1LL * k * a / g % m;
    }
    if (b == k) {
        return add;
    }
    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i) {
        an = 1LL * an * a % m;
    }
    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q < n; ++q) {
        vals[cur] = q;
        cur = 1LL * a * cur % m;
    }
    for (int p = 1, cur = k; p <= n; ++p) {
        cur = 1LL * cur * an % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
```

```
    }
    return -1;
}
```

6.14 Quadratic Residue

```
int Jacobi(int a, int m) {
  int s = 1;
  for (; m > 1; ) {
    a %= m;
    if (a == 0) return 0;
    const int r = __builtin_ctz(a);
    if ((r & 1) && ((m + 2) & 4)) s = -s;
    a >>= r;
    if (a & m & 2) s = -s;
    swap(a, m);
  }
  return s;
}
int QuadraticResidue(int a, int p) {
  if (p == 2) return a & 1;
  const int jc = Jacobi(a, p);
  if (jc == 0 || jc == -1) return jc;
  int b, d;
  for (; ; ) {
    b = rand() % p;
    d = (1LL * b * b + p - a) % p;
    if (Jacobi(d, p) == -1) break;
  }
  int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
  for (int e = (p + 1) >> 1; e; e >>= 1) {
    if (e & 1) {
      tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1 * f1 % p)) % p
          ;
      g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
      g0 = tmp;
    }
    tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) % p;
    f1 = (2LL * f0 * f1) % p;
    f0 = tmp;
  }
  return g0;
}
```

6.15 Characteristic Polynomial

```
vector<vector<int>> Hessenberg(const vector<vector<int>> &A) {
  int N = A.size();
  vector<vector<int>> H = A;
  for (int i = 0; i < N - 2; ++i) {
    if (!H[i + 1][i]) {
      for (int j = i + 2; j < N; ++j) {
        if (H[j][i]) {
          for (int k = i; k < N; ++k) swap(H[i + 1][k], H[j][k
              ]);
          for (int k = 0; k < N; ++k) swap(H[k][i + 1], H[k][j
              ]);
          break;
        }
      }
    }
    if (!H[i + 1][i]) continue;
    int val = fpow(H[i + 1][i], kP - 2);
    for (int j = i + 2; j < N; ++j) {
      int coef = 1LL * val * H[j][i] % kP;
      for (int k = i; k < N; ++k) H[j][k] = (H[j][k] + 1LL * H[
          i + 1][k] * (kP - coef)) % kP;
      for (int k = 0; k < N; ++k) H[k][i + 1] = (H[k][i + 1] +
          1LL * H[k][j] * coef) % kP;
    }
  }
  return H;
}
vector<int> CharacteristicPoly(const vector<vector<int>> &A) {
  int N = A.size();
  auto H = Hessenberg(A);
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) H[i][j] = kP - H[i][j];
  }
  vector<vector<int>> P(N + 1, vector<int>(N + 1));
  P[0][0] = 1;
  for (int i = 1; i <= N; ++i) {
    P[i][0] = 0;
    for (int j = 1; j <= i; ++j) P[i][j] = P[i - 1][j - 1];
    int val = 1;
    for (int j = i - 1; j >= 0; --j) {
      int coef = 1LL * val * H[j][i - 1] % kP;
      for (int k = 0; k <= j; ++k) P[i][k] = (P[i][k] + 1LL * P
          [j][k] * coef) % kP;
      if (j) val = 1LL * val * (kP - H[j][j - 1]) % kP;
    }
  }
  if (N & 1) {
    for (int i = 0; i <= N; ++i) P[N][i] = kP - P[N][i];
  }
  return P[N];
}
```

6.16 Linear Sieve Related

```
vector<int> minp(N + 1), primes, mobius(N + 1);
mobius[1] = 1;
for (int i = 2; i <= N; i++) {
    if (!minp[i]) {
        primes.push_back(i);
        minp[i] = i;
        mobius[i] = -1;
    }
    for (int p : primes) {
        if (p > N / i) {
            break;
        }
        minp[p * i] = p;
        mobius[p * i] = -mobius[i];
        if (i % p == 0) {
            mobius[p * i] = 0;
            break;
        }
    }
}
```

6.17 Partition Function

```
void Build(int n) {
  // P[i] = the number of ways to represent i as the sum of a
      non-decreasing sequence.
  vector<pair<int, int>> g = {{0, 0}};
  for (int i = 1; g.back().second <= n; ++i) {
    g.emplace_back(i % 2 ? 1 : kMod - 1, i * (3 * i - 1) / 2);
    g.emplace_back(i % 2 ? 1 : kMod - 1, i * (3 * i + 1) / 2);
  }
  P[0] = P[1] = 1;
  for (int i = 2; i <= n; ++i) {
    for (auto it : g) {
      if (i < it.second) continue;
      (P[i] += 1LL * P[i - it.second] * it.first % kMod) %=
          kMod;
    }
  }
}
```

6.18 De Bruijn Sequence

```
int res[kN], aux[kN], a[kN], sz;
void Rec(int t, int p, int n, int k) {
  if (t > n) {
    if (n % p == 0)
      for (int i = 1; i <= p; ++i) res[sz++] = aux[i];
  } else {
    aux[t] = aux[t - p];
    Rec(t + 1, p, n, k);
    for (aux[t] = aux[t - p] + 1; aux[t] < k; ++aux[t]) Rec(t +
         1, t, n, k);
  }
}
int DeBruijn(int k, int n) {
  // return cyclic string of length k^n such that every string
      of length n using k character appears as a substring.
  if (k == 1) return res[0] = 0, 1;
  fill(aux, aux + k * n, 0);
  return sz = 0, Rec(1, 1, n, k), sz;
}
```

6.19 Extended GCD

```
array<i64, 3> extgcd(i64 a, i64 b) {
  if (!b) { return {a, 1, 0}; }
  auto [g, x, y] = extgcd(b, a % b);
  return {g, y, x - a / b * y};
}
```

6.20 Euclidean Algorithms

- $m = \lfloor \frac{an+b}{c} \rfloor$

$$g(a,b,c,n) = \sum_{i=0}^{n} i \lfloor \frac{ai+b}{c} \rfloor$$
$$= \begin{cases} \lfloor \frac{a}{c} \rfloor \cdot \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor \cdot \frac{n(n+1)}{2} & \\ +g(a \bmod c, b \bmod c, c, n), & a \ge c \vee b \ge c \\ 0, & n < 0 \vee a = 0 \\ \frac{1}{2} \cdot (n(n+1)m - f(c, c-b-1, a, m-1) & \\ -h(c, c-b-1, a, m-1)), & \text{otherwise} \end{cases}$$

$$h(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$$
$$= \begin{cases} \lfloor \frac{a}{c} \rfloor^2 \cdot \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor^2 \cdot (n+1) & \\ +\lfloor \frac{a}{c} \rfloor \cdot \lfloor \frac{b}{c} \rfloor \cdot n(n+1) & \\ +h(a \bmod c, b \bmod c, c, n) & \\ +2\lfloor \frac{a}{c} \rfloor \cdot g(a \bmod c, b \bmod c, c, n) & \\ +2\lfloor \frac{b}{c} \rfloor \cdot f(a \bmod c, b \bmod c, c, n), & a \ge c \vee b \ge c \\ 0, & n < 0 \vee a = 0 \\ nm(m+1) - 2g(c, c-b-1, a, m-1) & \\ -2f(c, c-b-1, a, m-1) - f(a,b,c,n), & \text{otherwise} \end{cases}$$

6.21 Floor Sum

```
// \sum {i = 0} {n} floor((a * i + b) / c)
i64 floorSum(i64 a, i64 b, i64 c, i64 n) {
    if (n < 0) { return 0; }
    if (n == 0) { return b / c; }
    if (a == 0) { return b / c * (n + 1); }
    i64 res = 0;
    if (a >= c) { res += a / c * n * (n + 1) / 2, a %= c; }
    if (b >= c) { res += b / c * (n + 1), b %= c; }
    i64 m = (a * n + b) / c;
    return res + n * m - (m == 0 ? 0 : floorSum(c, c - b - 1, a
        , m - 1));
}
```

6.22 Chinese Remainder Theorem

```
long long crt(vector<int> mod, vector<int> a) {
  long long mult = mod[0];
  int n = (int)mod.size();
  long long res = a[0];
  for (int i = 1; i < n; ++i) {
    long long d, x, y;
    tie(d, x, y) = extgcd(mult, mod[i] * 1ll);
    if ((a[i] - res) % d) return -1;
    long long new_mult = mult / __gcd(mult, 1ll * mod[i]) * mod
        [i];
    res += x * ((a[i] - res) / d) % new_mult * mult % new_mult;
    mult = new_mult;
    ((res %= mult) += mult) %= mult;
  }
  return res;
}
```

6.23 Theorem

6.23.1 Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G, where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i,j) in G.

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

6.23.2 Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i,j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{rank(D)}{2}$ is the maximum matching on G.

6.23.3 Cayley's Formula

- Given a degree sequence $d_1, d_2, \ldots, d_n$ for each *labeled* vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ spanning trees.
- Let $T_{n,k}$ be the number of *labeled* forests on n vertices with k components, such that vertex $1, 2, \ldots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

6.23.4 Erdős–Gallai Theorem

A sequence of non-negative integers $d_1 \ge d_2 \ge \ldots \ge d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \ldots + d_n$ is even and

$$\sum_{i=1}^{k} d_i \le k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

holds for all $1 \le k \le n$.

7 Dynamic Programming

7.1 Dynamic Convex Hull

```
struct Line {
    // kx + b
  mutable i64 k, b, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(i64 x) const { return p < x; }
};
struct DynamicConvexHullMax : multiset<Line, less<>> {
  // (for doubles, use INF = 1/.0, div(a,b) = a/b)
  static constexpr i64 INF = numeric_limits<i64>::max();
  i64 div(i64 a, i64 b) {
      // floor
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = INF, 0;
    if (x->k == y->k) x->p = x->b > y->b ? INF : -INF;
    else x->p = div(y->b - x->b, x->k - y->k);
    return x->p >= y->p;
  }
  void add(i64 k, i64 b) {
    auto z = insert({k, 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));
  }
  i64 query(i64 x) {
      if (empty()) {
          return -INF;
      }
    auto l = *lower_bound(x);
    return l.k * x + l.b;
  }
};
```

7.2 $1D/1D$ Convex Optimization

```
struct segment {
  int i, l, r;
  segment(int a, int b, int c): i(a), l(b), r(c) {}
};
inline long long f(int l, int r) { return dp[l] + w(l + 1, r);
    }
void solve() {
  dp[0] = 0ll;
  deque&lt;segment&gt; deq; deq.push_back(segment(0, 1, n));
  for (int i = 1; i <= n; ++i) {
    dp[i] = f(deq.front().i, i);
    while (deq.size() && deq.front().r < i + 1) deq.pop_front()
        ;
    deq.front().l = i + 1;
    segment seg = segment(i, i + 1, n);
    while (deq.size() && f(i, deq.back().l) < f(deq.back().i,
        deq.back().l)) deq.pop_back();
    if (deq.size()) {
      int d = 1048576, c = deq.back().l;
      while (d >>= 1) if (c + d <= deq.back().r) {
        if (f(i, c + d) > f(deq.back().i, c + d)) c += d;
      }
      deq.back().r = c; seg.l = c + 1;
    }
    if (seg.l <= n) deq.push_back(seg);
  }
}
```

7.3 Conditon

7.3.1 Totally Monotone (Concave/Convex)

$\forall i < i', j < j',\ B[i][j] \le B[i'][j] \implies B[i][j'] \le B[i'][j']$
$\forall i < i', j < j',\ B[i][j] \ge B[i'][j] \implies B[i][j'] \ge B[i'][j']$

7.3.2 Monge Condition (Concave/Convex)

$\forall i < i', j < j'$, $B[i][j] + B[i'][j'] \geq B[i][j'] + B[i'][j]$
$\forall i < i', j < j'$, $B[i][j] + B[i'][j'] \leq B[i][j'] + B[i'][j]$

7.3.3 Optimal Split Point

If

$$B[i][j] + B[i+1][j+1] \geq B[i][j+1] + B[i+1][j]$$

then

$$H_{i,j-1} \leq H_{i,j} \leq H_{i+1,j}$$

8 Geometry

8.1 Basic

```cpp
bool same(double a, double b) { return abs(a - b) < eps; }

struct P {
  double x, y;
  P() : x(0), y(0) {}
  P(double x, double y) : x(x), y(y) {}
  P operator + (P b) { return P(x + b.x, y + b.y); }
  P operator - (P b) { return P(x - b.x, y - b.y); }
  P operator * (double b) { return P(x * b, y * b); }
  P operator / (double b) { return P(x / b, y / b); }
  double operator * (P b) { return x * b.x + y * b.y; }
  double operator ^ (P b) { return x * b.y - y * b.x; }
  double abs() { return hypot(x, y); }
  P unit() { return *this / abs(); }
  P rot(double o) {
    double c = cos(o), s = sin(o);
    return P(c * x - s * y, s * x + c * y);
  }
  double angle() { return atan2(y, x); }
};

struct L {
  // ax + by + c = 0
  double a, b, c, o;
  P pa, pb;
  L() : a(0), b(0), c(0), o(0), pa(), pb() {}
  L(P pa, P pb) : a(pa.y - pb.y), b(pb.x - pa.x), c(pa ^ pb), o
      (atan2(-a, b)), pa(pa), pb(pb) {}
  P project(P p) { return pa + (pb - pa).unit() * ((pb - pa) *
      (p - pa) / (pb - pa).abs()); }
  P reflect(P p) { return p + (project(p) - p) * 2; }
  double get_ratio(P p) { return (p - pa) * (pb - pa) / ((pb -
      pa).abs() * (pb - pa).abs()); }
};

bool SegmentIntersect(P p1, P p2, P p3, P p4) {
  if (max(p1.x, p2.x) < min(p3.x, p4.x) || max(p3.x, p4.x) <
      min(p1.x, p2.x)) return false;
  if (max(p1.y, p2.y) < min(p3.y, p4.y) || max(p3.y, p4.y) <
      min(p1.y, p2.y)) return false;
  return sign((p3 - p1) ^ (p4 - p1)) * sign((p3 - p2) ^ (p4 -
      p2)) <= 0 &&
    sign((p1 - p3) ^ (p2 - p3)) * sign((p1 - p4) ^ (p2 - p4))
        <= 0;
}

bool parallel(L x, L y) { return same(x.a * y.b, x.b * y.a); }

P Intersect(L x, L y) { return P(-x.b * y.c + x.c * y.b, x.a *
    y.c - x.c * y.a) / (-x.a * y.b + x.b * y.a); }
```

8.2 KD Tree

```cpp
namespace kdt {
int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn], yl[maxn], yr[
    maxn];
point p[maxn];
int build(int l, int r, int dep = 0) {
  if (l == r) return -1;
  function<bool(const point &, const point &)> f = [dep](const
      point &a, const point &b) {
    if (dep & 1) return a.x < b.x;
    else return a.y < b.y;
  };
  int m = (l + r) >> 1;
  nth_element(p + l, p + m, p + r, f);
  xl[m] = xr[m] = p[m].x;
  yl[m] = yr[m] = p[m].y;
  lc[m] = build(l, m, dep + 1);
  if (~lc[m]) {
    xl[m] = min(xl[m], xl[lc[m]]);
    xr[m] = max(xr[m], xr[lc[m]]);
    yl[m] = min(yl[m], yl[lc[m]]);
    yr[m] = max(yr[m], yr[lc[m]]);
  }
  rc[m] = build(m + 1, r, dep + 1);
  if (~rc[m]) {
    xl[m] = min(xl[m], xl[rc[m]]);
    xr[m] = max(xr[m], xr[rc[m]]);
    yl[m] = min(yl[m], yl[rc[m]]);
    yr[m] = max(yr[m], yr[rc[m]]);
  }
  return m;
}
bool bound(const point &q, int o, long long d) {
  double ds = sqrt(d + 1.0);
  if (q.x < xl[o] - ds || q.x > xr[o] + ds ||
      q.y < yl[o] - ds || q.y > yr[o] + ds) return false;
  return true;
}
long long dist(const point &a, const point &b) {
  return (a.x - b.x) * 1ll * (a.x - b.x) +
    (a.y - b.y) * 1ll * (a.y - b.y);
}
void dfs(const point &q, long long &d, int o, int dep = 0) {
  if (!bound(q, o, d)) return;
  long long cd = dist(p[o], q);
  if (cd != 0) d = min(d, cd);
  if ((dep & 1) && q.x < p[o].x || !(dep & 1) && q.y < p[o].y)
      {
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
  } else {
    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
  }
}
void init(const vector<point> &v) {
  for (int i = 0; i < v.size(); ++i) p[i] = v[i];
  root = build(0, v.size());
}
long long nearest(const point &q) {
  long long res = 1e18;
  dfs(q, res, root);
  return res;
}
}}
```

8.3 Delaunay Triangulation

Description: Fast Delaunay triangulation assuming no duplicates and not all points collinear (in latter case, result will be empty). Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in ccw order. Each circumcircle will contain none of the input points. If coordinates are ints at most B then T should be large enough to support ints on the order of B^4. We don't need double in Point if the coordinates are integers.

```cpp
namespace delaunay {
// Not equal to any other points.
const Point kA(inf, inf);
bool InCircle(Point p, Point a, Point b, Point c) {
  a = a - p;
  b = b - p;
  c = c - p;
  __int128 x = __int128(a.Norm()) * (b ^ c) + __int128(b.Norm()
      ) * (c ^ a) +
    __int128(c.Norm()) * (a ^ b);
  return x * Sign((b - a) ^ (c - a)) > 0;
}
struct Quad {
  bool mark;
  Quad *o, *rot;
  Point p;
  Quad(Point p) : mark(false), o(nullptr), rot(nullptr), p(p)
      {}
  Point F() { return r()->p; }
  Quad* r() { return rot->rot; }
  Quad* prev() { return rot->o->rot; }
  Quad* next() { return r()->prev(); }
};
Quad* MakeEdge(Point orig, Point dest) {
  Quad* q[4] = {new Quad(orig), new Quad(kA), new Quad(dest),
      new Quad(kA)};
  for (int i = 0; i < 4; ++i) {
    q[i]->o = q[-i & 3];
    q[i]->rot = q[(i + 1) & 3];
  }
  return q[0];
}
```

```cpp
void Splice(Quad* a, Quad* b) {
  swap(a->o->rot->o, b->o->rot->o);
  swap(a->o, b->o);
}
Quad* Connect(Quad* a, Quad* b) {
  Quad* q = MakeEdge(a->F(), b->p);
  Splice(q, a->next());
  Splice(q->r(), b);
  return q;
}
pair<Quad*, Quad*> Dfs(const vector<Point>& s, int l, int r) {
  if (r - l <= 3) {
    Quad *a = MakeEdge(s[l], s[l + 1]), *b = MakeEdge(s[l + 1],
         s[r - 1]);
    if (r - l == 2) return {a, a->r()};
    Splice(a->r(), b);
    auto side = (s[l + 1] - s[l]) ^ (s[l + 2] - s[l]);
    Quad* c = side ? Connect(b, a) : nullptr;
    return make_pair(side < 0 ? c->r() : a, side < 0 ? c : b->r
        ());
  }
  int m = (l + r) >> 1;
  auto [ra, a] = Dfs(s, l, m);
  auto [b, rb] = Dfs(s, m, r);
  while (((a->F() - b->p) ^ (a->p - b->p)) < 0 && (a = a->next
      ()) ||
     ((b->F() - a->p) ^ (b->p - a->p)) > 0 && (b = b->r()->o))
        ;
  Quad* base = Connect(b->r(), a);
  auto Valid = [&](Quad* e) {
    return ((base->F() - e->F()) ^ (base->p - e->F())) > 0;
  };
  if (a->p == ra->p) ra = base->r();
  if (b->p == rb->p) rb = base;
  while (true) {
    Quad* lc = base->r()->o;
    if (Valid(lc)) {
      while (InCircle(lc->o->F(), base->F(), base->p, lc->F()))
          {
        Quad* t = lc->o;
        Splice(lc, lc->prev());
        Splice(lc->r(), lc->r()->prev());
        lc = t;
      }
    }
    Quad* rc = base->prev();
    if (Valid(rc)) {
      while (InCircle(rc->prev()->F(), base->F(), base->p, rc->
          F())) {
        Quad* t = rc->prev();
        Splice(rc, rc->prev());
        Splice(rc->r(), rc->r()->prev());
        rc = t;
      }
    }
    if (!Valid(lc) && !Valid(rc)) break;
    if (!Valid(lc) || (Valid(rc) && InCircle(rc->F(), rc->p, lc
        ->F(), lc->p))) {
      base = Connect(rc, base->r());
    } else {
      base = Connect(base->r(), lc->r());
    }
  }
  return make_pair(ra, rb);
}
vector<array<Point, 3>> Triangulate(vector<Point> pts) {
  sort(pts.begin(), pts.end());
  if (pts.size() < 2) return {};
  Quad* e = Dfs(pts, 0, pts.size()).first;
  vector<Quad*> q = {e};
  while (((e->F() - e->o->F()) ^ (e->p - e->o->F())) < 0) e = e
      ->o;
  auto Add = [&]() {
    Quad* c = e;
    do {
      c->mark = true;
      pts.push_back(c->p);
      q.push_back(c->r());
      c = c->next();
    } while (c != e);
  };
  Add();
  pts.clear();
  int ptr = 0;
  while (ptr < q.size()) {
    if (!(e = q[ptr++])->mark) Add();
  }
  vector<array<Point, 3>> res(pts.size() / 3);
  for (int i = 0; i < pts.size(); ++i) res[i / 3][i % 3] = pts[
      i];
  return res;
}
}  // namespace delaunay
```

8.4 Voronoi Diagram

Description: Vertices in Voronoi Diagram are circumcenters of triangles in the Delaunay Triangulation.

```cpp
int gid(P &p) {
  auto it = ptoid.find(p);
  if (it == ptoid.end()) return -1;
  return it->second;
}
L make_line(P p, L l) {
  P d = l.pb - l.pa; d = d.rot(pi / 2);
  P m = (l.pa + l.pb) / 2;
  l = L(m, m + d);
  if (((l.pb - l.pa) ^ (p - l.pa)) < 0) l = L(m + d, m);
  return l;
}
double calc_ans(int i) {
  vector<P> ps = HPI(ls[i]);
  double rt = 0;
  for (int i = 0; i < (int)ps.size(); ++i) {
    rt += (ps[i] ^ ps[(i + 1) % ps.size()]);
  }
  return abs(rt) / 2;
}
void solve() {
  for (int i = 0; i < n; ++i) ops[i] = ps[i], ptoid[ops[i]] = i
      ;
  random_shuffle(ps, ps + n);
  build(n, ps);
  for (auto *t : triang) {
    int z[3] = {gid(t->p[0]), gid(t->p[1]), gid(t->p[2])};
    for (int i = 0; i < 3; ++i) for (int j = 0; j < 3; ++j) if
        (i != j && z[i] != -1 && z[j] != -1) {
      L l(t->p[i], t->p[j]);
      ls[z[i]].push_back(make_line(t->p[i], l));
    }
  }
  vector<P> tb = convex(vector<P>(ps, ps + n));
  for (auto &p : tb) isinf[gid(p)] = true;
  for (int i = 0; i < n; ++i) {
    if (isinf[i]) cout << -1 << '\n';
    else cout << fixed << setprecision(12) << calc_ans(i) << '\
        n';
  }
}
```

8.5 Sector Area

```cpp
// calc area of sector which include a, b
double SectorArea(P a, P b, double r) {
  double o = atan2(a.y, a.x) - atan2(b.y, b.x);
  while (o <= 0) o += 2 * pi;
  while (o >= 2 * pi) o -= 2 * pi;
  o = min(o, 2 * pi - o);
  return r * r * o / 2;
}
```

8.6 Half Plane Intersection

```cpp
bool jizz(L l1,L l2,L l3){
  P p=Intersect(l2,l3);
  return ((l1.pb-l1.pa)^(p-l1.pa))<-eps;
}

bool cmp(const L &a,const L &b){
  return same(a.o,b.o)?(((b.pb-b.pa)^(a.pb-b.pa))>eps):a.o<b.o;
}

// availble area for L l is (l.pb-l.pa)^(p-l.pa)>0
vector<P> HPI(vector<L> &ls){
  sort(ls.begin(),ls.end(),cmp);
  vector<L> pls(1,ls[0]);
  for(int i=0;i<(int)ls.size();++i)if(!same(ls[i].o,pls.back().
      o))pls.push_back(ls[i]);
  deque<int> dq; dq.push_back(0); dq.push_back(1);
#define meow(a,b,c) while(dq.size()>1u && jizz(pls[a],pls[b],
    pls[c]))
  for(int i=2;i<(int)pls.size();++i){
    meow(i,dq.back(),dq[dq.size()-2])dq.pop_back();
    meow(i,dq[0],dq[1])dq.pop_front();
```

```
    dq.push_back(i);
  }
  meow(dq.front(),dq.back(),dq[dq.size()-2])dq.pop_back();
  meow(dq.back(),dq[0],dq[1])dq.pop_front();
  if(dq.size()<3u)return vector<P>(); // no solution or
       solution is not a convex
  vector<P> rt;
  for(int i=0;i<(int)dq.size();++i)rt.push_back(Intersect(pls[
       dq[i]],pls[dq[(i+1)%dq.size()]]));
  return rt;
}
```

8.7 Triangle Center

```
Point TriangleCircumCenter(Point a, Point b, Point c) {
  Point res;
  double a1 = atan2(b.y - a.y, b.x - a.x) + pi / 2;
  double a2 = atan2(c.y - b.y, c.x - b.x) + pi / 2;
  double ax = (a.x + b.x) / 2;
  double ay = (a.y + b.y) / 2;
  double bx = (c.x + b.x) / 2;
  double by = (c.y + b.y) / 2;
  double r1 = (sin(a2) * (ax - bx) + cos(a2) * (by - ay)) / (
       sin(a1) * cos(a2) - sin(a2) * cos(a1));
  return Point(ax + r1 * cos(a1), ay + r1 * sin(a1));
}

Point TriangleMassCenter(Point a, Point b, Point c) {
  return (a + b + c) / 3.0;
}

Point TriangleOrthoCenter(Point a, Point b, Point c) {
  return TriangleMassCenter(a, b, c) * 3.0 -
       TriangleCircumCenter(a, b, c) * 2.0;
}

Point TriangleInnerCenter(Point a, Point b, Point c) {
  Point res;
  double la = len(b - c);
  double lb = len(a - c);
  double lc = len(a - b);
  res.x = (la * a.x + lb * b.x + lc * c.x) / (la + lb + lc);
  res.y = (la * a.y + lb * b.y + lc * c.y) / (la + lb + lc);
  return res;
}
```

8.8 Polygon Center

```
Point BaryCenter(vector<Point> &p, int n) {
  Point res(0, 0);
  double s = 0.0, t;
  for (int i = 1; i < p.size() - 1; i++) {
    t = Cross(p[i] - p[0], p[i + 1] - p[0]) / 2;
    s += t;
    res.x += (p[0].x + p[i].x + p[i + 1].x) * t;
    res.y += (p[0].y + p[i].y + p[i + 1].y) * t;
  }
  res.x /= (3 * s);
  res.y /= (3 * s);
  return res;
}
```

8.9 Maximum Triangle

```
double ConvexHullMaxTriangleArea(Point p[], int res[], int
     chnum) {
  double area = 0, tmp;
  res[chnum] = res[0];
  for (int i = 0, j = 1, k = 2; i < chnum; i++) {
    while (fabs(Cross(p[res[j]] - p[res[i]], p[res[(k + 1) %
         chnum]] - p[res[i]])) > fabs(Cross(p[res[j]] - p[res[i
         ]], p[res[k]] - p[res[i]]))) k = (k + 1) % chnum;
    tmp = fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] - p[res[i
         ]]));
    if (tmp > area) area = tmp;
    while (fabs(Cross(p[res[(j + 1) % chnum]] - p[res[i]], p[
         res[k]] - p[res[i]])) > fabs(Cross(p[res[j]] - p[res[i
         ]], p[res[k]] - p[res[i]]))) j = (j + 1) % chnum;
    tmp = fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] - p[res[i
         ]]));
    if (tmp > area) area = tmp;
  }
  return area / 2;
}
```

8.10 Point in Polygon

```
int pip(vector<P> ps, P p) {
  int c = 0;
  for (int i = 0; i < ps.size(); ++i) {
    int a = i, b = (i + 1) % ps.size();
    L l(ps[a], ps[b]);
    P q = l.project(p);
    if ((p - q).abs() < eps && l.inside(q)) return 1;
    if (same(ps[a].y, ps[b].y) && same(ps[a].y, p.y)) continue;
    if (ps[a].y > ps[b].y) swap(a, b);
    if (ps[a].y <= p.y && p.y < ps[b].y && p.x <= ps[a].x + (ps
         [b].x - ps[a].x) / (ps[b].y - ps[a].y) * (p.y - ps[a].
         y)) ++c;
  }
  return (c & 1) * 2;
}
```

8.11 Circle

```
struct C {
  P c;
  double r;
  C(P c = P(0, 0), double r = 0) : c(c), r(r) {}
};

vector<P> Intersect(C a, C b) {
  if (a.r > b.r) swap(a, b);
  double d = (a.c - b.c).abs();
  vector<P> p;
  if (same(a.r + b.r, d)) p.push_back(a.c + (b.c - a.c).unit()
       * a.r);
  else if (a.r + b.r > d && d + a.r >= b.r) {
    double o = acos((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d)
         );
    P i = (b.c - a.c).unit();
    p.push_back(a.c + i.rot(o) * a.r);
    p.push_back(a.c + i.rot(-o) * a.r);
  }
  return p;
}
double IntersectArea(C a, C b) {
  if (a.r > b.r) swap(a, b);
  double d = (a.c - b.c).abs();
  if (d >= a.r + b.r - eps) return 0;
  if (d + a.r <= b.r + eps) return sq(a.r) * acos(-1);
  double p = acos((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d));
  double q = acos((sq(b.r) + sq(d) - sq(a.r)) / (2 * b.r * d));
  return p * sq(a.r) + q * sq(b.r) - a.r * d * sin(p);
}
// remove second level if to get points for line (defalut:
     segment)
vector<P> CircleCrossLine(P a, P b, P o, double r) {
  double x = b.x - a.x, y = b.y - a.y, A = sq(x) + sq(y), B = 2
       * x * (a.x - o.x) + 2 * y * (a.y - o.y);
  double C = sq(a.x - o.x) + sq(a.y - o.y) - sq(r), d = B * B -
       4 * A * C;
  vector<P> t;
  if (d >= -eps) {
    d = max(0., d);
    double i = (-B - sqrt(d)) / (2 * A);
    double j = (-B + sqrt(d)) / (2 * A);
    if (i - 1.0 <= eps && i >= -eps) t.emplace_back(a.x + i * x
         , a.y + i * y);
    if (j - 1.0 <= eps && j >= -eps) t.emplace_back(a.x + j * x
         , a.y + j * y);
  }
  return t;
}
// calc area intersect by circle with radius r and triangle OAB
double AreaOfCircleTriangle(P a, P b, double r) {
  bool ina = a.abs() < r, inb = b.abs() < r;
  auto p = CircleCrossLine(a, b, P(0, 0), r);
  if (ina) {
    if (inb) return abs(a ^ b) / 2;
    return SectorArea(b, p[0], r) + abs(a ^ p[0]) / 2;
  }
  if (inb) return SectorArea(p[0], a, r) + abs(p[0] ^ b) / 2;
  if (p.size() == 2u) return SectorArea(a, p[0], r) +
       SectorArea(p[1], b, r) + abs(p[0] ^ p[1]) / 2;
  else return SectorArea(a, b, r);
}
// for any triangle
double AreaOfCircleTriangle(vector<P> ps, double r) {
  double ans = 0;
  for (int i = 0; i < 3; ++i) {
    int j = (i + 1) % 3;
```

```
    double o = atan2(ps[i].y, ps[i].x) - atan2(ps[j].y, ps[j].x
        );
    if (o >= pi) o = o - 2 * pi;
    if (o <= -pi) o = o + 2 * pi;
    ans += AreaOfCircleTriangle(ps[i], ps[j], r) * (o >= 0 ? 1
        : -1);
  }
  return abs(ans);
}
```

8.12 Tangent of Circles and Points to Circle

```
vector<L> tangent(C a, C b) {
#define Pij \
  P i = (b.c - a.c).unit() * a.r, j = P(i.y, -i.x);\
  z.emplace_back(a.c + i, a.c + i + j);
#define deo(I,J) \
  double d = (a.c - b.c).abs(), e = a.r I b.r, o = acos(e / d)
      ;\
  P i = (b.c - a.c).unit(), j = i.rot(o), k = i.rot(-o);\
  z.emplace_back(a.c + j * a.r, b.c J j * b.r);\
  z.emplace_back(a.c + k * a.r, b.c J k * b.r);
  if (a.r < b.r) swap(a, b);
  vector<L> z;
  if ((a.c - b.c).abs() + b.r < a.r) return z;
  else if (same((a.c - b.c).abs() + b.r, a.r)) { Pij; }
  else {
    deo(-,+);
    if (same(d, a.r + b.r)) { Pij; }
    else if (d > a.r + b.r) { deo(+,-); }
  }
  return z;
}

vector<L> tangent(C c, P p) {
  vector<L> z;
  double d = (p - c.c).abs();
  if (same(d, c.r)) {
    P i = (p - c.c).rot(pi / 2);
    z.emplace_back(p, p + i);
  } else if (d > c.r) {
    double o = acos(c.r / d);
    P i = (p - c.c).unit(), j = i.rot(o) * c.r, k = i.rot(-o) *
        c.r;
    z.emplace_back(c.c + j, p);
    z.emplace_back(c.c + k, p);
  }
  return z;
}
```

8.13 Area of Union of Circles

```
vector<pair<double, double>> CoverSegment(C &a, C &b) {
  double d = (a.c - b.c).abs();
  vector<pair<double, double>> res;
  if (same(a.r + b.r, d)) ;
  else if (d <= abs(a.r - b.r) + eps) {
    if (a.r < b.r) res.emplace_back(0, 2 * pi);
  } else if (d < abs(a.r + b.r) - eps) {
    double o = acos((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d)
        ), z = (b.c - a.c).angle();
    if (z < 0) z += 2 * pi;
    double l = z - o, r = z + o;
    if (l < 0) l += 2 * pi;
    if (r > 2 * pi) r -= 2 * pi;
    if (l > r) res.emplace_back(l, 2 * pi), res.emplace_back(0,
         r);
    else res.emplace_back(l, r);
  }
  return res;
}
double CircleUnionArea(vector<C> c) { // circle should be
    identical
  int n = c.size();
  double a = 0, w;
  for (int i = 0; w = 0, i < n; ++i) {
    vector<pair<double, double>> s = {{2 * pi, 9}}, z;
    for (int j = 0; j < n; ++j) if (i != j) {
      z = CoverSegment(c[i], c[j]);
      for (auto &e : z) s.push_back(e);
    }
    sort(s.begin(), s.end());
    auto F = [&] (double t) { return c[i].r * (c[i].r * t + c[i
        ].c.x * sin(t) - c[i].c.y * cos(t)); };
    for (auto &e : s) {
      if (e.first > w) a += F(e.first) - F(w);
      w = max(w, e.second);
    }
  }
  return a * 0.5;
}
```

8.14 Minimun Distance of 2 Polygons

```
// p, q is convex
double TwoConvexHullMinDist(Point P[], Point Q[], int n, int m)
    {
  int YMinP = 0, YMaxQ = 0;
  double tmp, ans = 999999999;
  for (i = 0; i < n; ++i) if(P[i].y < P[YMinP].y) YMinP = i;
  for (i = 0; i < m; ++i) if(Q[i].y > Q[YMaxQ].y) YMaxQ = i;
  P[n] = P[0], Q[m] = Q[0];
  for (int i = 0; i < n; ++i) {
    while (tmp = Cross(Q[YMaxQ + 1] - P[YMinP + 1], P[YMinP] -
        P[YMinP + 1]) > Cross(Q[YMaxQ] - P[YMinP + 1], P[YMinP
        ] - P[YMinP + 1])) YMaxQ = (YMaxQ + 1) % m;
    if (tmp < 0) ans = min(ans, PointToSegDist(P[YMinP], P[
        YMinP + 1], Q[YMaxQ]));
    else ans = min(ans, TwoSegMinDist(P[YMinP], P[YMinP + 1], Q
        [YMaxQ], Q[YMaxQ + 1]));
    YMinP = (YMinP + 1) % n;
  }
  return ans;
}
```

8.15 2D Convex Hull

```
bool operator<(const P &a, const P &b) {
  return same(a.x, b.x) ? a.y < b.y : a.x < b.x;
}
bool operator>(const P &a, const P &b) {
  return same(a.x, b.x) ? a.y > b.y : a.x > b.x;
}

#define crx(a, b, c) ((b - a) ^ (c - a))

vector<P> convex(vector<P> ps) {
  vector<P> p;
  sort(ps.begin(), ps.end(), [&] (P a, P b) { return same(a.x,
      b.x) ? a.y < b.y : a.x < b.x; });
  for (int i = 0; i < ps.size(); ++i) {
    while (p.size() >= 2 && crx(p[p.size() - 2], ps[i], p[p.
        size() - 1]) >= 0) p.pop_back();
    p.push_back(ps[i]);
  }
  int t = p.size();
  for (int i = (int)ps.size() - 2; i >= 0; --i) {
    while (p.size() > t && crx(p[p.size() - 2], ps[i], p[p.size
        () - 1]) >= 0) p.pop_back();
    p.push_back(ps[i]);
  }
  p.pop_back();
  return p;
}

int sgn(double x) { return same(x, 0) ? 0 : x > 0 ? 1 : -1; }

P isLL(P p1, P p2, P q1, P q2) {
  double a = crx(q1, q2, p1), b = -crx(q1, q2, p2);
  return (p1 * b + p2 * a) / (a + b);
}

struct CH {
  int n;
  vector<P> p, u, d;
  CH() {}
  CH(vector<P> ps) : p(ps) {
    n = ps.size();
    rotate(p.begin(), min_element(p.begin(), p.end()), p.end())
        ;
    auto t = max_element(p.begin(), p.end());
    d = vector<P>(p.begin(), next(t));
    u = vector<P>(t, p.end()); u.push_back(p[0]);
  }
  int find(vector<P> &v, P d) {
    int l = 0, r = v.size();
    while (l + 5 < r) {
      int L = (l * 2 + r) / 3, R = (l + r * 2) / 3;
      if (v[L] * d > v[R] * d) r = R;
      else l = L;
    }
    int x = l;
    for (int i = l + 1; i < r; ++i) if (v[i] * d > v[x] * d) x
        = i;
    return x;
```

```
  }
  int findFarest(P v) {
    if (v.y > 0 || v.y == 0 && v.x > 0) return ((int)d.size() -
         1 + find(u, v)) % p.size();
    return find(d, v);
  }
  P get(int l, int r, P a, P b) {
    int s = sgn(crx(a, b, p[l % n]));
    while (l + 1 < r) {
      int m = (l + r) >> 1;
      if (sgn(crx(a, b, p[m % n])) == s) l = m;
      else r = m;
    }
    return isLL(a, b, p[l % n], p[(l + 1) % n]);
  }
  vector<P> getLineIntersect(P a, P b) {
    int X = findFarest((b - a).rot(pi / 2));
    int Y = findFarest((a - b).rot(pi / 2));
    if (X > Y) swap(X, Y);
    if (sgn(crx(a, b, p[X])) * sgn(crx(a, b, p[Y])) < 0) return
         {get(X, Y, a, b), get(Y, X + n, a, b)};
    return {}; // tangent case falls here
  }
  void update_tangent(P q, int i, int &a, int &b) {
    if (sgn(crx(q, p[a], p[i])) > 0) a = i;
    if (sgn(crx(q, p[b], p[i])) < 0) b = i;
  }
  void bs(int l, int r, P q, int &a, int &b) {
    if (l == r) return;
    update_tangent(q, l % n, a, b);
    int s = sgn(crx(q, p[l % n], p[(l + 1) % n]));
    while (l + 1 < r) {
      int m = (l + r) >> 1;
      if (sgn(crx(q, p[m % n], p[(m + 1) % n])) == s) l = m;
      else r = m;
    }
    update_tangent(q, r % n, a, b);
  }
  int x = l;
  for (int i = l + 1; i < r; ++i) if (v[i] * d > v[x] * d) x =
       i;
  return x;
}
int findFarest(P v) {
  if (v.y > 0 || v.y == 0 && v.x > 0) return ((int)d.size() - 1
       + find(u, v)) % p.size();
  return find(d, v);
}
P get(int l, int r, P a, P b) {
  int s = sgn(crx(a, b, p[l % n]));
  while (l + 1 < r) {
    int m = (l + r) >> 1;
    if (sgn(crx(a, b, p[m % n])) == s) l = m;
    else r = m;
  }
  return isLL(a, b, p[l % n], p[(l + 1) % n]);
}
vector<P> getIS(P a, P b) {
  int X = findFarest((b - a).spin(pi / 2));
  int Y = findFarest((a - b).spin(pi / 2));
  if (X > Y) swap(X, Y);
  if (sgn(crx(a, b, p[X])) * sgn(crx(a, b, p[Y])) < 0) return {
      get(X, Y, a, b), get(Y, X + n, a, b)};
  return {};
}
void update_tangent(P q, int i, int &a, int &b) {
  if (sgn(crx(q, p[a], p[i])) > 0) a = i;
  if (sgn(crx(q, p[b], p[i])) < 0) b = i;
}
void bs(int l, int r, P q, int &a, int &b) {
  if (l == r) return;
  update_tangent(q, l % n, a, b);
  int s = sgn(crx(q, p[l % n], p[(l + 1) % n]));
  while (l + 1 < r) {
    int m = (l + r) >> 1;
    if (sgn(crx(q, p[m % n], p[(m + 1) % n])) == s) l = m;
    else r = m;
  }
  update_tangent(q, r % n, a, b);
}
bool contain(P p) {
  if (p.x < d[0].x || p.x > d.back().x) return 0;
  auto it = lower_bound(d.begin(), d.end(), P(p.x, -1e12));
  if (it->x == p.x) {
    if (it->y > p.y) return 0;
  } else if (crx(*prev(it), *it, p) < -eps) return 0;
  it = lower_bound(u.begin(), u.end(), P(p.x, 1e12), greater<P
      >());
  if (it->x == p.x) {
    if (it->y < p.y) return 0;
  } else if (crx(*prev(it), *it, p) < -eps) return 0;
  return 1;
}
bool get_tangent(P p, int &a, int &b) { // b -> a
  if (contain(p)) return 0;
  a = b = 0;
  int i = lower_bound(d.begin(), d.end(), p) - d.begin();
  bs(0, i, p, a, b);
  bs(i, d.size(), p, a, b);
  i = lower_bound(u.begin(), u.end(), p, greater<P>()) - u.
       begin();
  bs((int)d.size() - 1, (int)d.size() - 1 + i, p, a, b);
  bs((int)d.size() - 1 + i, (int)d.size() - 1 + u.size(), p, a,
       b);
  return 1;
}
};
```

8.16 3D Convex Hull

```
double absvol(const P a,const P b,const P c,const P d) {
  return abs(((b-a)^(c-a))*(d-a))/6;
}
struct convex3D {
  static const int maxn=1010;
  struct T{
    int a,b,c;
    bool res;
    T(){}
    T(int a,int b,int c,bool res=1):a(a),b(b),c(c),res(res){}
  };
  int n,m;
  P p[maxn];
  T f[maxn*8];
  int id[maxn][maxn];
  bool on(T &t,P &q){
    return ((p[t.c]-p[t.b])^(p[t.a]-p[t.b]))*(q-p[t.a])>eps;
  }
  void meow(int q,int a,int b){
    int g=id[a][b];
    if(f[g].res){
      if(on(f[g],p[q]))dfs(q,g);
      else{
        id[q][b]=id[a][q]=id[b][a]=m;
        f[m++]=T(b,a,q,1);
      }
    }
  }
  void dfs(int p,int i){
    f[i].res=0;
    meow(p,f[i].b,f[i].a);
    meow(p,f[i].c,f[i].b);
    meow(p,f[i].a,f[i].c);
  }
  void operator()(){
    if(n<4)return;
    if([&](){
        for(int i=1;i<n;++i)if(abs(p[0]-p[i])>eps)return swap(p
             [1],p[i]),0;
        return 1;
        }() || [&](){
        for(int i=2;i<n;++i)if(abs((p[0]-p[i])^(p[1]-p[i]))>eps
             )return swap(p[2],p[i]),0;
        return 1;
        }() || [&](){
        for(int i=3;i<n;++i)if(abs(((p[1]-p[0])^(p[2]-p[0]))*(p
             [i]-p[0]))>eps)return swap(p[3],p[i]),0;
        return 1;
        }())return;
    for(int i=0;i<4;++i){
      T t((i+1)%4,(i+2)%4,(i+3)%4,1);
      if(on(t,p[i]))swap(t.b,t.c);
      id[t.a][t.b]=id[t.b][t.c]=id[t.c][t.a]=m;
      f[m++]=t;
    }
    for(int i=4;i<n;++i)for(int j=0;j<m;++j)if(f[j].res && on(f
         [j],p[i])){
      dfs(i,j);
      break;
    }
    int mm=m; m=0;
    for(int i=0;i<mm;++i)if(f[i].res)f[m++]=f[i];
  }
```

```
  bool same(int i,int j){
    return !(absvol(p[f[i].a],p[f[i].b],p[f[i].c],p[f[j].a])>
         eps || absvol(p[f[i].a],p[f[i].b],p[f[i].c],p[f[j].b])
         >eps || absvol(p[f[i].a],p[f[i].b],p[f[i].c],p[f[j].c
         ])>eps);
  }
  int faces(){
    int r=0;
    for(int i=0;i<m;++i){
      int iden=1;
      for(int j=0;j<i;++j)if(same(i,j))iden=0;
      r+=iden;
    }
    return r;
  }
} tb;
```

8.17 Closest Pair

```
double closest_pair(int l, int r) {
  // p should be sorted increasingly according to the x-
       coordinates.
  if (l == r) return 1e9;
  if (r - l == 1) return dist(p[l], p[r]);
  int m = (l + r) >> 1;
  double d = min(closest_pair(l, m), closest_pair(m + 1, r));
  vector<int> vec;
  for (int i = m; i >= l && fabs(p[m].x - p[i].x) < d; --i) vec
       .push_back(i);
  for (int i = m + 1; i <= r && fabs(p[m].x - p[i].x) < d; ++i)
       vec.push_back(i);
  sort(vec.begin(), vec.end(), [&](int a, int b) { return p[a].
       y < p[b].y; });
  for (int i = 0; i < vec.size(); ++i) {
    for (int j = i + 1; j < vec.size() && fabs(p[vec[j]].y - p[
         vec[i]].y) < d; ++j) {
      d = min(d, dist(p[vec[i]], p[vec[j]]));
    }
  }
  return d;
}
```

9 Miscellaneous

9.1 Cactus

```
// a component contains no articulation point, so P2 is a
     component
// resulting bct is rooted
struct BlockCutTree {
    int n, square = 0, cur = 0;
    vector<int> low, dfn, stk;
    vector<vector<int>> g, bct;
    BlockCutTree(int n) : n(n), low(n), dfn(n, -1), g(n), bct(n
         ) {}
    void build() { dfs(0); }
    void addEdge(int u, int v) { g[u].push_back(v), g[v].
         push_back(u); }
    void dfs(int u) {
        low[u] = dfn[u] = cur++;
        stk.push_back(u);
        for (auto v : g[u]) {
            if (dfn[v] == -1) {
                dfs(v);
                low[u] = min(low[u], low[v]);
                if (low[v] == dfn[u]) {
                    bct.emplace_back();
                    int x;
                    do {
                        x = stk.back();
                        stk.pop_back();
                        bct.back().push_back(x);
                    } while (x != v);
                    bct[u].push_back(n + square);
                    square++;
                }
            } else {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }
};
```

9.2 Dancing Links

```
namespace dlx {
int lt[maxn], rg[maxn], up[maxn], dn[maxn], cl[maxn], rw[maxn],
     bt[maxn], s[maxn], head, sz, ans;
void init(int c) {
  for (int i = 0; i < c; ++i) {
    up[i] = dn[i] = bt[i] = i;
    lt[i] = i == 0 ? c : i - 1;
    rg[i] = i == c - 1 ? c : i + 1;
    s[i] = 0;
  }
  rg[c] = 0, lt[c] = c - 1;
  up[c] = dn[c] = -1;
  head = c, sz = c + 1;
}
void insert(int r, const vector<int> &col) {
  if (col.empty()) return;
  int f = sz;
  for (int i = 0; i < (int)col.size(); ++i) {
    int c = col[i], v = sz++;
    dn[bt[c]] = v;
    up[v] = bt[c], bt[c] = v;
    rg[v] = (i + 1 == (int)col.size() ? f : v + 1);
    rw[v] = r, cl[v] = c;
    ++s[c];
    if (i > 0) lt[v] = v - 1;
  }
  lt[f] = sz - 1;
}
void remove(int c) {
  lt[rg[c]] = lt[c], rg[lt[c]] = rg[c];
  for (int i = dn[c]; i != c; i = dn[i]) {
    for (int j = rg[i]; j != i; j = rg[j])
      up[dn[j]] = up[j], dn[up[j]] = dn[j], --s[cl[j]];
  }
}
void restore(int c) {
  for (int i = up[c]; i != c; i = up[i]) {
    for (int j = lt[i]; j != i; j = lt[j])
      ++s[cl[j]], up[dn[j]] = j, dn[up[j]] = j;
  }
  lt[rg[c]] = c, rg[lt[c]] = c;
}
// Call dlx::make after inserting all rows.
void make(int c) {
  for (int i = 0; i < c; ++i)
    dn[bt[i]] = i, up[i] = bt[i];
}
void dfs(int dep) {
  if (dep >= ans) return;
  if (rg[head] == head) return ans = dep, void();
  if (dn[rg[head]] == rg[head]) return;
  int c = rg[head];
  int w = c;
  for (int x = c; x != head; x = rg[x]) if (s[x] < s[w]) w = x;
  remove(w);
  for (int i = dn[w]; i != w; i = dn[i]) {
    for (int j = rg[i]; j != i; j = rg[j]) remove(cl[j]);
    dfs(dep + 1);
    for (int j = lt[i]; j != i; j = lt[j]) restore(cl[j]);
  }
  restore(w);
}
int solve() {
  ans = 1e9, dfs(0);
  return ans;
}}
```

9.3 Offline Dynamic MST

```
int cnt[maxn], cost[maxn], st[maxn], ed[maxn];
pair<int, int> qr[maxn];
// qr[i].first = id of edge to be changed, qr[i].second =
     weight after operation
// cnt[i] = number of operation on edge i
// call solve(0, q - 1, v, 0), where v contains edges i such
     that cnt[i] == 0

void contract(int l, int r, vector<int> v, vector<int> &x,
     vector<int> &y) {
  sort(v.begin(), v.end(), [&](int i, int j) {
      if (cost[i] == cost[j]) return i < j;
      return cost[i] < cost[j];
      });
  djs.save();
  for (int i = l; i <= r; ++i) djs.merge(st[qr[i].first], ed[qr
       [i].first]);
```

```
  for (int i = 0; i < (int)v.size(); ++i) {
    if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
      x.push_back(v[i]);
      djs.merge(st[v[i]], ed[v[i]]);
    }
  }
  djs.undo();
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i) djs.merge(st[x[i]],
     ed[x[i]]);
  for (int i = 0; i < (int)v.size(); ++i) {
    if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
      y.push_back(v[i]);
      djs.merge(st[v[i]], ed[v[i]]);
    }
  }
  djs.undo();
}

void solve(int l, int r, vector<int> v, long long c) {
  if (l == r) {
    cost[qr[l].first] = qr[l].second;
    if (st[qr[l].first] == ed[qr[l].first]) {
      printf("%lld\n", c);
      return;
    }
    int minv = qr[l].second;
    for (int i = 0; i < (int)v.size(); ++i) minv = min(minv,
        cost[v[i]]);
    printf("%lld\n", c + minv);
    return;
  }
  int m = (l + r) >> 1;
  vector<int> lv = v, rv = v;
  vector<int> x, y;
  for (int i = m + 1; i <= r; ++i) {
    cnt[qr[i].first]--;
    if (cnt[qr[i].first] == 0) lv.push_back(qr[i].first);
  }
  contract(l, m, lv, x, y);
  long long lc = c, rc = c;
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i) {
    lc += cost[x[i]];
    djs.merge(st[x[i]], ed[x[i]]);
  }
  solve(l, m, y, lc);
  djs.undo();
  x.clear(), y.clear();
  for (int i = m + 1; i <= r; ++i) cnt[qr[i].first]++;
  for (int i = l; i <= m; ++i) {
    cnt[qr[i].first]--;
    if (cnt[qr[i].first] == 0) rv.push_back(qr[i].first);
  }
  contract(m + 1, r, rv, x, y);
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i) {
    rc += cost[x[i]];
    djs.merge(st[x[i]], ed[x[i]]);
  }
  solve(m + 1, r, y, rc);
  djs.undo();
  for (int i = l; i <= m; ++i) cnt[qr[i].first]++;
}
```

9.4 Manhattan Distance MST

```
void solve(int n) {
  init();
  vector<int> v(n), ds;
  for (int i = 0; i < n; ++i) {
    v[i] = i;
    ds.push_back(x[i] - y[i]);
  }
  sort(ds.begin(), ds.end());
  ds.resize(unique(ds.begin(), ds.end()) - ds.begin());
  sort(v.begin(), v.end(), [&](int i, int j) { return x[i] == x
     [j] ? y[i] > y[j] : x[i] > x[j]; });
  int j = 0;
  for (int i = 0; i < n; ++i) {
    int p = lower_bound(ds.begin(), ds.end(), x[v[i]] - y[v[i
        ]]) - ds.begin() + 1;
    pair<int, int> q = query(p);
    // query return prefix minimum
    if (~q.second) add_edge(v[i], q.second);
    add(p, make_pair(x[v[i]] + y[v[i]], v[i]));
  }
}

void make_graph() {
  solve(n);
  for (int i = 0; i < n; ++i) swap(x[i], y[i]);
  solve(n);
  for (int i = 0; i < n; ++i) x[i] = -x[i];
  solve(n);
  for (int i = 0; i < n; ++i) swap(x[i], y[i]);
  solve(n);
}
```

9.5 Matroid Intersection

- $x \to y$ if $S - \{x\} \cup \{y\} \in I_1$ with $cost(\{y\})$.
- $source \to y$ if $S \cup \{y\} \in I_1$ with $cost(\{y\})$.
- $y \to x$ if $S - \{x\} \cup \{y\} \in I_2$ with $-cost(\{y\})$.
- $y \to sink$ if $S \cup \{y\} \in I_2$ with $-cost(\{y\})$.

Augmenting path is shortest path from source to sink.