KTH Royal Institute of Technology

Omogen Heap

Simon Lindholm, Johan Sannemo, Mårten Wiman

2024-11-14

graphs (1)

BellmanFord.cpp

Description: desc

e48dec, 74 lines

```
// https://cp-algorithms.com/graph/bellman_ford.html

typedef pair<int, int> edge;

const int INF = numeric_limits<int>::max() / 3;

tuple<bool, vector<int>, vector<int>> bellman_ford(const vector
     <vector<edge>> &g, int s) {
    size_t n = g.size();
    vector<int> prio(n, INF);
    prio[s] = 0;
    vector<int> pred(n, -1);
    bool was_changed = true;
    for (int k = 0; k < n; k++) {
        was_changed = false;
        for (int u = 0; u < n; u++) {
            for (auto [v, cost] : g[u]) {
                if (prio[v] > prio[u] + cost) {
                    prio[v] = prio[u] + cost;
                    pred[v] = u;
                    was_changed = true;
                }
            }
        }
        if (!was_changed)
            break;
    }
    // was_changed is true iff graph has a negative cycle
    return {was_changed, prio, pred};
}
```

```
vector<int> find_negative_cycle(const vector<vector<edge>> &g)
     {
    size_t n = g.size();
    vector<int> pred(n, -1);
    vector<int> prio(n, INF);
    prio[0] = 0;
    int last = 0;
    for (int k = 0; k < n; k++) {
        last = -1;
        for (int u = 0; u < n; u++) {
            for (auto [v, cost] : g[u]) {
                if (prio[v] > prio[u] + cost) {
                    prio[v] = prio[u] + cost;
                    pred[v] = u;
                    last = v;
                }
            }
        }
        if (last == -1)
            return {};
    }

    vector<int> path(n);
    vector<int> pos(n, -1);
    for (int i = 0;; i++) {
        if (pos[last] != -1)
            return vector<int>(path.rend() - i, path.rend() -
                 pos[last]);
        path[i] = last;
        pos[last] = i;
        last = pred[last];
    }
}

int main() {
    vector<vector<edge>> g(4);
    g[0].emplace_back(1, 1);
    g[1].emplace_back(0, 1);
    g[1].emplace_back(2, 1);
    g[2].emplace_back(3, -10);
    g[3].emplace_back(1, 1);

    vector<int> cycle = find_negative_cycle(g);
    for (int u : cycle)
        cout << u << " ";
}
```

CycleDetection.cpp

Description: desc

bf7a7c, 51 lines

```
int dfs(const vector<vector<int>> &graph, int u, vector<int> &
     color, vector<int> &next) {
    color[u] = 1;
    for (int v : graph[u]) {
        next[u] = v;
        if (color[v] == 0) {
            int cycleStart = dfs(graph, v, color, next);
            if (cycleStart != -1) {
                return cycleStart;
            }
        } else if (color[v] == 1) {
            return v;
        }
    }
    color[u] = 2;
    return -1;
}

vector<int> find_cycle(const vector<vector<int>> &graph) {
    int n = graph.size();
    vector<int> color(n);
    vector<int> next(n);
    for (int u = 0; u < n; u++) {
        if (color[u] != 0)
            continue;
        int cycleStart = dfs(graph, u, color, next);
        if (cycleStart != -1) {
            vector<int> cycle;
            cycle.push_back(cycleStart);
            for (int i = next[cycleStart]; i != cycleStart; i =
                 next[i]) {
                cycle.push_back(i);
            }
            cycle.push_back(cycleStart);
            return cycle;
        }
    }
    return {};
}

// usage example
int main() {
    vector<vector<int>> graph{{1}, {2}, {0}};
    auto cycle = find_cycle(graph);
    cout << cycle.size() << endl;
    for (int x : cycle)
        cout << x << " ";
    cout << endl;

    graph = {{1}, {2}, {}};
    cycle = find_cycle(graph);
    cout << cycle.size() << endl;
}
```

Dijkstra.cpp

Description: desc

9a8489, 76 lines

```
typedef pair<int, int> edge;
typedef pair<int, int> item;

// https://cp-algorithms.com/graph/dijkstra_sparse.html

// O(E*log(V)) time and O(E) memory
tuple<vector<int>, vector<int>> dijkstra_heap(const vector<
     vector<edge>> &g, int s) {
    size_t n = g.size();
    vector<int> prio(n, numeric_limits<int>::max());
    vector<int> pred(n, -1);
    priority_queue<item, vector<item>, greater<>> q;
    q.emplace(prio[s] = 0, s);

    while (!q.empty()) {
        auto [d, u] = q.top();
        q.pop();

        if (d != prio[u])
            continue;

        for (auto [v, len] : g[u]) {
            int nprio = prio[u] + len;
            if (prio[v] > nprio) {
                prio[v] = nprio;
                pred[v] = u;
                q.emplace(nprio, v);
            }
        }
    }
```

```
    return {prio, pred};
}

// O(E*log(V)) time and O(V) memory
tuple<vector<int>, vector<int>> dijkstra_set(const vector<
     vector<edge>> &g, int s) {
    size_t n = g.size();
    vector<int> prio(n, numeric_limits<int>::max());
    vector<int> pred(n, -1);
    set<item> q;
    q.emplace(prio[s] = 0, s);

    while (!q.empty()) {
        int u = q.begin()->second;
        q.erase(q.begin());

        for (auto [v, len] : g[u]) {
            int nprio = prio[u] + len;
            if (prio[v] > nprio) {
                q.erase({prio[v], v});
                prio[v] = nprio;
                pred[v] = u;
                q.emplace(prio[v], v);
            }
        }
    }

    return {prio, pred};
}

int main() {
    vector<vector<edge>> g(3);
    g[0].emplace_back(1, 10);
    g[1].emplace_back(2, -5);
    g[0].emplace_back(2, 8);

    auto [prio1, pred1] = dijkstra_heap(g, 0);
    auto [prio2, pred2] = dijkstra_set(g, 0);

    for (int x : prio1)
        cout << x << " ";
    cout << endl;

    for (int x : prio2)
        cout << x << " ";
    cout << endl;
}
```

DijkstraCustomHeap.cpp

Description: desc

9e60b8, 118 lines

```
const int maxnodes = 200'000;
const int maxedges = 1000'000;

// graph
int edges;
int last[maxnodes], head[maxedges], previous[maxedges], len[
     maxedges];
int prio[maxnodes], pred[maxnodes];

void reset_graph() {
    fill(last, last + maxnodes, -1);
    edges = 0;
}

void add_edge(int u, int v, int length) {
    head[edges] = v;
    len[edges] = length;
    previous[edges] = last[u];
    last[u] = edges++;
}

// heap
int h[maxnodes];
int pos2Id[maxnodes];
int id2Pos[maxnodes];
int hsize;

void hswap(int i, int j) {
    swap(h[i], h[j]);
    swap(pos2Id[i], pos2Id[j]);
    swap(id2Pos[pos2Id[i]], id2Pos[pos2Id[j]]);
}

void move_up(int pos) {
    while (pos > 0) {
        int parent = (pos - 1) >> 1;
        if (h[pos] >= h[parent]) {
            break;
        }
        hswap(pos, parent);
        pos = parent;
    }
}

void add(int id, int prio) {
    h[hsize] = prio;
    pos2Id[hsize] = id;
    id2Pos[id] = hsize;
    move_up(hsize++);
}

void decrease_value(int id, int prio) {
    int pos = id2Pos[id];
    h[pos] = prio;
    move_up(pos);
}

void move_down(int pos) {
    while (pos < (hsize >> 1)) {
        int child = 2 * pos + 1;
        if (child + 1 < hsize && h[child + 1] < h[child]) {
            ++child;
        }
        if (h[pos] <= h[child]) {
            break;
        }
        hswap(pos, child);
        pos = child;
    }
}

int remove_min() {
    int res = pos2Id[0];
    int lastNode = h[--hsize];
    if (hsize > 0) {
        h[0] = lastNode;
        int id = pos2Id[hsize];
        id2Pos[id] = 0;
        pos2Id[0] = id;
        move_down(0);
    }
    return res;
}

void dijkstra(int s) {
    fill(pred, pred + maxnodes, -1);
    fill(prio, prio + maxnodes, numeric_limits<int>::max());
    prio[s] = 0;
    hsize = 0;
    add(s, prio[s]);

    while (hsize) {
        int u = remove_min();
        for (int e = last[u]; e >= 0; e = previous[e]) {
            int v = head[e];
            int nprio = prio[u] + len[e];
            if (prio[v] > nprio) {
                if (prio[v] == numeric_limits<int>::max())
                    add(v, nprio);
                else
                    decrease_value(v, nprio);
                prio[v] = nprio;
                pred[v] = u;
            }
        }
    }
}

int main() {
    reset_graph();
    add_edge(0, 1, 10);
    add_edge(1, 2, -5);
    add_edge(0, 2, 8);

    dijkstra(0);

    for (int i = 0; i < 3; i++)
        cout << prio[i] << endl;
}
```

GlobalMinCutStoerWagner.cpp

Description: desc

0ceec0, 52 lines

```
// https://en.wikipedia.org/wiki/Stoer%E2%80%93Wagner_algorithm
     in O(V^3)

pair<int, vector<int>> min_cut(vector<vector<int>> &cap) {
    int best_cap = numeric_limits<int>::max();
    vector<int> best_cut;
    int n = cap.size();
    vector<vector<int>> v(n);
    for (int i = 0; i < n; ++i)
        v[i].push_back(i);
    vector<int> w(n);
    vector<bool> exist(n, true);
    vector<bool> in_a(n);
    for (int ph = 0; ph < n - 1; ++ph) {
        fill(in_a.begin(), in_a.end(), false);
        fill(w.begin(), w.end(), 0);
        for (int it = 0, prev; it < n - ph; ++it) {
            int sel = -1;
            for (int i = 0; i < n; ++i)
                if (exist[i] && !in_a[i] && (sel == -1 || w[i]
                     > w[sel])) {
                    sel = i;
                }
            if (it == n - ph - 1) {
                if (w[sel] < best_cap) {
                    best_cap = w[sel];
                    best_cut = v[sel];
                }
                v[prev].insert(v[prev].end(), v[sel].begin(), v
                     [sel].end());
                for (int i = 0; i < n; ++i) {
                    cap[i][prev] += cap[sel][i];
                    cap[prev][i] += cap[sel][i];
```

```
                }
                exist[sel] = false;
            } else {
                in_a[sel] = true;
                for (int i = 0; i < n; ++i)
                    w[i] += cap[sel][i];
                prev = sel;
            }
        }
    }
    return {best_cap, best_cut};
}

// usage example
int main() {
    vector<vector<int>> capacity{{0, 1, 1, 0}, {1, 0, 1, 1}, {
        1, 1, 0, 1}, {0, 1, 1, 0}};
    auto [cap, cut] = min_cut(capacity);
    cout << cap << endl;
    for (int v : cut)
        cout << v << " ";
    cout << endl;
}
```

GomoryHuTree.cpp

Description: desc

"max_flow_dinic.h" aa9622, 31 lines

```
vector<vector<pair<int, int>>> gomory_hu_tree(max_flow_dinic &
    flow) {
    int n = flow.graph.size();
    vector<vector<pair<int, int>>> t(n, vector<pair<int, int
        >>());
    vector<int> p(n);
    for (int i = 1; i < n; i++) {
        flow.clear_flow();
        int f = flow.max_flow(i, p[i]);
        vector<bool> cut = flow.min_cut();
        for (int j = i + 1; j < n; j++)
            if (cut[j] == cut[i] && p[j] == p[i])
                p[j] = i;
        t[p[i]].emplace_back(i, f);
    }
    return t;
}

// usage example
int main() {
    int capacity[][3] = {{0, 3, 2}, {0, 0, 2}, {0, 0, 0}};
    int n = 3;
    max_flow_dinic flow(n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (capacity[i][j] != 0)
                flow.add_edge(i, j, capacity[i][j]);

    vector<vector<pair<int, int>>> t = gomory_hu_tree(flow);
    for (int u = 0; u < n; ++u)
        for (auto [v, f] : t[u])
            cout << u << " " << v << " " << f << endl;
}
```

LcaRmqSchieberVishkin.cpp

Description: desc

2f8390, 125 lines

```
#ifdef _MSC_VER
int __builtin_clz(unsigned x) {
    int bit = 31;
    while (bit >= 0 && (x & (1 << bit)) == 0)
        --bit;
    return 31 - bit;
}
#endif

const int MAX_NODES = 200'000;
int parent[MAX_NODES];
unsigned pre_order[MAX_NODES];
unsigned I[MAX_NODES];
int head[MAX_NODES];
unsigned A[MAX_NODES];
unsigned Time;

unsigned lowest_one_bit(unsigned x) {
    return x & -x;
}

unsigned highest_one_bit(unsigned x) {
    return x ? 1u << (31 - __builtin_clz(x)) : 0;
}

void dfs1(const vector<vector<int>> &tree, int u, int p) {
    parent[u] = p;
    I[u] = pre_order[u] = Time++;
    for (int v : tree[u]) {
        if (v == p)
            continue;
        dfs1(tree, v, u);
        if (lowest_one_bit(I[u]) < lowest_one_bit(I[v])) {
            I[u] = I[v];
        }
    }
    head[I[u]] = u;
}

void dfs2(const vector<vector<int>> &tree, int u, int p,
    unsigned up) {
    A[u] = up | lowest_one_bit(I[u]);
    for (int v : tree[u]) {
        if (v == p)
            continue;
        dfs2(tree, v, u, A[u]);
    }
}

// initialization in O(n)
void init_lca(const vector<vector<int>> &tree, int root) {
    Time = 0;
    dfs1(tree, root, -1);
    dfs2(tree, root, -1, 0);
}

int enter_into_strip(int x, int hz) {
    if (lowest_one_bit(I[x]) == hz)
        return x;
    int hw = highest_one_bit(A[x] & (hz - 1));
    return parent[head[(I[x] & -hw) | hw]];
}

// lca in O(1)
int lca(int x, int y) {
    int hb = I[x] == I[y] ? lowest_one_bit(I[x]) :
        highest_one_bit(I[x] ^ I[y]);
    int hz = lowest_one_bit(A[x] & A[y] & -hb);
    int ex = enter_into_strip(x, hz);
    int ey = enter_into_strip(y, hz);
    return pre_order[ex] < pre_order[ey] ? ex : ey;
}

void init_rmq(const vector<int> &values) {
    // build Cartesian Tree
    int n = values.size();
    int root = 0;
    vector<int> p(n, -1);
    for (int i = 1; i < n; i++) {
        int prev = i - 1;
        int next = -1;
        while (values[prev] > values[i] && p[prev] != -1) {
            next = prev;
            prev = p[prev];
        }
        if (values[prev] > values[i]) {
            p[prev] = i;
            root = i;
        } else {
            p[i] = prev;
            if (next != -1) {
                p[next] = i;
            }
        }
    }
    vector<vector<int>> tree(n);
    for (int i = 0; i < n; ++i)
        if (p[i] != -1)
            tree[p[i]].push_back(i);
    init_lca(tree, root);
}

// random test
int main() {
    mt19937 rng(1);
    for (int step = 0; step < 1000; ++step) {
        int n = uniform_int_distribution<int>(1, 10)(rng);
        vector<int> v(n);
        for (int i = 0; i < n; ++i) {
            v[i] = uniform_int_distribution<int>(0, 5)(rng);
        }
        int a = uniform_int_distribution<int>(0, n - 1)(rng);
        int b = uniform_int_distribution<int>(0, n - 1)(rng);
        if (a > b)
            swap(a, b);
        init_rmq(v);
        int res1 = v[lca(a, b)];
        int res2 = *min_element(v.begin() + a, v.begin() + b +
            1);
        if (res1 != res2) {
            for (int i = 0; i < n; ++i)
                cout << v[i] << " ";
            cout << endl;
            cout << a << " " << b << " - " << res1 << " " <<
                res2 << endl;
            assert(res1 != res2);
        }
    }
}
```

MaxBipartiteMatchingEv.cpp

Description: desc

164dfe, 42 lines

```
// https://en.wikipedia.org/wiki/Matching_(graph_theory)#
    In_unweighted_bipartite_graphs
// time complexity: O(E * V)

bool findPath(const vector<vector<int>> &graph, int u1, vector<
    int> &matching, vector<bool> &vis) {
    vis[u1] = true;
    for (int v : graph[u1]) {
        int u2 = matching[v];
```

```
        if (u2 == -1 || (!vis[u2] && findPath(graph, u2,
             matching, vis))) {
            matching[v] = u1;
            return true;
        }
    }
    return false;
}

tuple<int, vector<int>> max_matching(const vector<vector<int>>
     &graph, int n2) {
    int n1 = graph.size();
    vector<int> matching(n2, -1);
    int matches = 0;
    for (int u = 0; u < n1; u++) {
        vector<bool> vis(n1);
        if (findPath(graph, u, matching, vis))
            ++matches;
    }
    return {matches, matching};
}

// usage example
int main() {
    vector<vector<int>> g(2);
    g[0].push_back(0);
    g[0].push_back(1);
    g[1].push_back(0);

    auto [max_matching_cardinality, mapping] = max_matching(g,
         2);

    cout << (2 == max_matching_cardinality) << endl;

    for (int x : mapping)
        cout << x << " ";
    cout << endl;
}
```

MaxBipartiteMatchingHopcroftKarpEsqrtv.cpp

Description: desc
Time: $\mathcal{O}(E * sqrt(V))$

227fd0, 71 lines

```
void bfs(const vector<vector<int>> &graph, vector<bool> &used,
     vector<int> &mapping, vector<int> &dist) {
    fill(dist.begin(), dist.end(), -1);
    size_t n1 = graph.size();
    vector<int> Q(n1);
    int sizeQ = 0;
    for (int u = 0; u < n1; ++u) {
        if (!used[u]) {
            Q[sizeQ++] = u;
            dist[u] = 0;
        }
    }
    for (int i = 0; i < sizeQ; i++) {
        int u1 = Q[i];
        for (int v : graph[u1]) {
            int u2 = mapping[v];
            if (u2 >= 0 && dist[u2] < 0) {
                dist[u2] = dist[u1] + 1;
                Q[sizeQ++] = u2;
            }
        }
    }
}

bool dfs(const vector<vector<int>> &graph, vector<bool> &vis,
     vector<bool> &used, vector<int> &matching,
         vector<int> &dist, int u1) {
    vis[u1] = true;
    for (int v : graph[u1]) {
        int u2 = matching[v];
        if (u2 < 0 || (!vis[u2] && dist[u2] == dist[u1] + 1 &&
             dfs(graph, vis, used, matching, dist, u2))) {
            matching[v] = u1;
            used[u1] = true;
            return true;
        }
    }
    return false;
}

tuple<int, vector<int>> max_matching(const vector<vector<int>>
     &graph, int n2) {
    vector<int> mapping(n2, -1);
    size_t n1 = graph.size();
    vector<int> dist(n1);
    vector<bool> used(n1);
    for (int res = 0;;) {
        bfs(graph, used, mapping, dist);
        vector<bool> vis(n1);
        int f = 0;
        for (int u = 0; u < n1; ++u)
            if (!used[u] && dfs(graph, vis, used, mapping, dist
                 , u))
                ++f;
        if (f == 0)
            return {res, mapping};
        res += f;
    }
}

// usage example
int main() {
    vector<vector<int>> g(3);
    g[0].push_back(0);
    g[0].push_back(1);
    g[1].push_back(1);
    g[2].push_back(1);

    auto [max_matching_cardinality, mapping] = max_matching(g,
         2);

    cout << (2 == max_matching_cardinality) << endl;

    for (int x : mapping)
        cout << x << " ";
    cout << endl;
}
```

MaxFlowDinic.cpp

Description: desc

"max_flow_dinic.h" eeb74b, 12 lines

```
// usage example
int main() {
    int capacity[][3] = {{0, 3, 2}, {0, 0, 2}, {0, 0, 0}};
    int n = 3;
    max_flow_dinic flow(n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (capacity[i][j] != 0)
                flow.add_edge(i, j, capacity[i][j]);

    cout << (4 == flow.max_flow(0, 2)) << endl;
}
```

MaxFlowDinic.h

Description: desc

cc3b04, 83 lines

```
// https://en.wikipedia.org/wiki/Dinic%27s_algorithm in O(V^2 *
     E)

struct Edge {
    int to, rev, cap, f;
};

struct max_flow_dinic {
    vector<vector<Edge>> graph;
    vector<int> dist;

    max_flow_dinic(int nodes) : graph(nodes), dist(nodes) {}

    void add_edge(int s, int t, int cap) {
        Edge a = {t, (int)graph[t].size(), cap, 0};
        Edge b = {s, (int)graph[s].size(), cap, 0};
        graph[s].emplace_back(a);
        graph[t].emplace_back(b);
    }

    bool dinic_bfs(int src, int dest) {
        fill(dist.begin(), dist.end(), -1);
        dist[src] = 0;
        vector<int> q(graph.size());
        int qt = 0;
        q[qt++] = src;
        for (int qh = 0; qh < qt; qh++) {
            int u = q[qh];
            for (auto &e : graph[u]) {
                int v = e.to;
                if (dist[v] < 0 && e.f < e.cap) {
                    dist[v] = dist[u] + 1;
                    q[qt++] = v;
                }
            }
        }
        return dist[dest] >= 0;
    }

    int dinic_dfs(vector<int> &ptr, int u, int dest, int f) {
        if (u == dest)
            return f;
        for (int &i = ptr[u]; i < (int)graph[u].size(); i++) {
            Edge &e = graph[u][i];
            if (e.cap <= e.f)
                continue;
            int v = e.to;
            if (dist[v] == dist[u] + 1) {
                int df = dinic_dfs(ptr, v, dest, min(f, e.cap -
                     e.f));
                if (df > 0) {
                    e.f += df;
                    graph[v][e.rev].f -= df;
                    return df;
                }
            }
        }
        return 0;
    }

    int max_flow(int src, int dest) {
        int flow = 0;
        while (dinic_bfs(src, dest)) {
            vector<int> ptr(graph.size());
            while (int delta = dinic_dfs(ptr, src, dest,
                 numeric_limits<int>::max()))
```

```
                flow += delta;
            }
        return flow;
    }

    // invoke after max_flow()
    vector<bool> min_cut() {
        vector<bool> cut(graph.size());
        for (size_t i = 0; i < cut.size(); ++i) {
            cut[i] = dist[i] != -1;
        }
        return cut;
    }

    void clear_flow() {
        for (auto &v : graph)
            for (Edge &e : v)
                e.f = 0;
    }
};
```

MaxFlowSimple.cpp

Description: Ford-Fulkerson
Time: $\mathcal{O}\left(V^2 * flow\right)$

6c2207, 26 lines

```
bool augment_path(vector<vector<int>> &cap, vector<bool> &vis,
     int i, int t) {
    if (i == t)
        return true;
    vis[i] = true;
    for (int j = 0; j < vis.size(); j++)
        if (!vis[j] && cap[i][j] > 0 && augment_path(cap, vis,
             j, t)) {
            --cap[i][j];
            ++cap[j][i];
            return true;
        }
    return false;
}

int maxFlow(vector<vector<int>> &cap, int s, int t) {
    for (int flow = 0;; ++flow) {
        vector<bool> vis(cap.size());
        if (!augment_path(cap, vis, s, t))
            return flow;
    }
}

// usage example
int main() {
    vector<vector<int>> capacity{{0, 1, 1, 0}, {1, 0, 1, 1}, {
         1, 1, 0, 1}, {0, 1, 1, 0}};
    cout << (2 == maxFlow(capacity, 0, 3)) << endl;
}
```

MaxGeneralMatchingEsqrtv.cpp

Description: desc

2a2780, 417 lines

```
// Taken from https://gist.github.com/min-25/
     aed29a23b004505d2094a5cddaf56ff9
// Tested: https://codeforces.com/contest/1198/submission
     /58096981

class MaximumMatching {
    /*
      Maximum Cardinality Matching in General Graphs.
      - O(\sqrt{n} m \log_{\max\{2, 1 + m/n\}} n) time
      - O(n + m) space

      Note: each vertex is 1-indexed.

      Ref:
        Harold N. Gabow,
        "The Weighted Matching Approach to Maximum Cardinality
             Matching" (2017)
        (https://arxiv.org/abs/1703.03998)
    */
public:
    struct Edge {
        int from, to;
    };
    static constexpr int Inf = 1 << 30;

private:
    enum Label {
        kInner = -1,  // should be < 0
        kFree = 0     // should be 0
    };
    struct Link {
        int from, to;
    };
    struct Log {
        int v, par;
    };

    struct LinkedList {
        LinkedList() {}

        LinkedList(int N, int M) : N(N), next(M) { clear(); }

        void clear() { head.assign(N, -1); }

        void push(int h, int u) { next[u] = head[h], head[h] =
             u; }

        int N;
        vector<int> head, next;
    };

    template <typename T>
    struct Queue {
        Queue() {}

        Queue(int N) : qh(0), qt(0), data(N) {}

        T operator[](int i) const { return data[i]; }

        void enqueue(int u) { data[qt++] = u; }

        int dequeue() { return data[qh++]; }

        bool empty() const { return qh == qt; }

        void clear() { qh = qt = 0; }

        int size() const { return qt; }

        int qh, qt;
        vector<T> data;
    };

    struct DisjointSetUnion {
        DisjointSetUnion() {}

        DisjointSetUnion(int N) : par(N) {
            for (int i = 0; i < N; ++i)
                par[i] = i;
        }

        int find(int u) { return par[u] == u ? u : (par[u] =
             find(par[u])); }

        void unite(int u, int v) {
            u = find(u), v = find(v);
            if (u != v)
                par[v] = u;
        }

        vector<int> par;
    };

public:
    MaximumMatching(int N, const vector<Edge> &in) : N(N), NH(N
         >> 1), ofs(N + 2, 0), edges(in.size() * 2) {
        for (auto &e : in)
            ofs[e.from + 1] += 1, ofs[e.to + 1] += 1;
        for (int i = 1; i <= N + 1; ++i)
            ofs[i] += ofs[i - 1];
        for (auto &e : in) {
            edges[ofs[e.from]++] = e;
            edges[ofs[e.to]++] = {e.to, e.from};
        }
        for (int i = N + 1; i > 0; --i)
            ofs[i] = ofs[i - 1];
        ofs[0] = 0;
    }

    int maximum_matching() {
        initialize();
        int match = 0;
        while (match * 2 + 1 < N) {
            reset_count();
            bool has_augmenting_path = do_edmonds_search();
            if (!has_augmenting_path)
                break;
            match += find_maximal();
            clear();
        }
        return match;
    }

private:
    void reset_count() {
        time_current_ = 0;
        time_augment_ = Inf;
        contract_count_ = 0;
        outer_id_ = 1;
        dsu_changelog_size_ = dsu_changelog_last_ = 0;
    }

    void clear() {
        que.clear();
        for (int u = 1; u <= N; ++u)
            potential[u] = 1;
        for (int u = 1; u <= N; ++u)
            dsu.par[u] = u;
        for (int t = time_current_; t <= N / 2; ++t)
            list.head[t] = -1;
        for (int u = 1; u <= N; ++u)
            blossom.head[u] = -1;
    }

    // first phase

    inline void grow(int x, int y, int z) {
        label[y] = kInner;
        potential[y] = time_current_;  // visited time
```

```
    link[z] = {x, y};
    label[z] = label[x];
    potential[z] = time_current_ + 1;
    que.enqueue(z);
}

void contract(int x, int y) {
    int bx = dsu.find(x), by = dsu.find(y);
    const int h = -(++contract_count_) + kInner;
    label[mate[bx]] = label[mate[by]] = h;
    int lca = -1;
    while (1) {
        if (mate[by] != 0)
            swap(bx, by);
        bx = lca = dsu.find(link[bx].from);
        if (label[mate[bx]] == h)
            break;
        label[mate[bx]] = h;
    }
    for (auto bv : {dsu.par[x], dsu.par[y]}) {
        for (; bv != lca; bv = dsu.par[link[bv].from]) {
            int mv = mate[bv];
            link[mv] = {x, y};
            label[mv] = label[x];
            potential[mv] = 1 + (time_current_ - potential[
                mv]) + time_current_;
            que.enqueue(mv);
            dsu.par[bv] = dsu.par[mv] = lca;
            dsu_changelog[dsu_changelog_last_++] = {bv, lca
                };
            dsu_changelog[dsu_changelog_last_++] = {mv, lca
                };
        }
    }
}

bool find_augmenting_path() {
    while (!que.empty()) {
        int x = que.dequeue(), lx = label[x], px =
            potential[x], bx = dsu.find(x);
        for (int eid = ofs[x]; eid < ofs[x + 1]; ++eid) {
            int y = edges[eid].to;
            if (label[y] > 0) {  // outer blossom/vertex
                int time_next = (px + potential[y]) >> 1;
                if (lx != label[y]) {
                    if (time_next == time_current_)
                        return true;
                    time_augment_ = min(time_next,
                        time_augment_);
                } else {
                    if (bx == dsu.find(y))
                        continue;
                    if (time_next == time_current_) {
                        contract(x, y);
                        bx = dsu.find(x);
                    } else if (time_next <= NH)
                        list.push(time_next, eid);
                }
            } else if (label[y] == kFree) {  // free vertex
                int time_next = px + 1;
                if (time_next == time_current_)
                    grow(x, y, mate[y]);
                else if (time_next <= NH)
                    list.push(time_next, eid);
            }
        }
    }
    return false;
}

bool adjust_dual_variables() {
    // Return true if the current matching is maximum.
    const int time_lim = min(NH + 1, time_augment_);
    for (++time_current_; time_current_ <= time_lim; ++
        time_current_) {
        dsu_changelog_size_ = dsu_changelog_last_;
        if (time_current_ == time_lim)
            break;
        bool updated = false;
        for (int h = list.head[time_current_]; h >= 0; h =
            list.next[h]) {
            auto &e = edges[h];
            int x = e.from, y = e.to;
            if (label[y] > 0) {
                // Case: outer --- (free => inner => outer)
                if (potential[x] + potential[y] != (
                    time_current_ << 1))
                    continue;
                if (dsu.find(x) == dsu.find(y))
                    continue;
                if (label[x] != label[y]) {
                    time_augment_ = time_current_;
                    return false;
                }
                contract(x, y);
                updated = true;
            } else if (label[y] == kFree) {
                grow(x, y, mate[y]);
                updated = true;
            }
        }
        list.head[time_current_] = -1;
        if (updated)
            return false;
    }
    return time_current_ > NH;
}

bool do_edmonds_search() {
    label[0] = kFree;
    for (int u = 1; u <= N; ++u) {
        if (mate[u] == 0) {
            que.enqueue(u);
            label[u] = u;  // component id
        } else
            label[u] = kFree;
    }
    while (1) {
        if (find_augmenting_path())
            break;
        bool maximum = adjust_dual_variables();
        if (maximum)
            return false;
        if (time_current_ == time_augment_)
            break;
    }
    for (int u = 1; u <= N; ++u) {
        if (label[u] > 0)
            potential[u] -= time_current_;
        else if (label[u] < 0)
            potential[u] = 1 + (time_current_ - potential[u
                ]);
    }
    return true;
}

// second phase

void rematch(int v, int w) {
    int t = mate[v];
    mate[v] = w;
    if (mate[t] != v)
        return;
    if (link[v].to == dsu.find(link[v].to)) {
        mate[t] = link[v].from;
        rematch(mate[t], t);
    } else {
        int x = link[v].from, y = link[v].to;
        rematch(x, y);
        rematch(y, x);
    }
}

bool dfs_augment(int x, int bx) {
    int px = potential[x], lx = label[bx];
    for (int eid = ofs[x]; eid < ofs[x + 1]; ++eid) {
        int y = edges[eid].to;
        if (px + potential[y] != 0)
            continue;
        int by = dsu.find(y), ly = label[by];
        if (ly > 0) {  // outer
            if (lx >= ly)
                continue;
            int stack_beg = stack_last_;
            for (int bv = by; bv != bx; bv = dsu.find(link[
                bv].from)) {
                int bw = dsu.find(mate[bv]);
                stack[stack_last_++] = bw;
                link[bw] = {x, y};
                dsu.par[bv] = dsu.par[bw] = bx;
            }
            while (stack_last_ > stack_beg) {
                int bv = stack[--stack_last_];
                for (int v = blossom.head[bv]; v >= 0; v =
                    blossom.next[v]) {
                    if (!dfs_augment(v, bx))
                        continue;
                    stack_last_ = stack_beg;
                    return true;
                }
            }
        } else if (ly == kFree) {
            label[by] = kInner;
            int z = mate[by];
            if (z == 0) {
                rematch(x, y);
                rematch(y, x);
                return true;
            }
            int bz = dsu.find(z);
            link[bz] = {x, y};
            label[bz] = outer_id_++;
            for (int v = blossom.head[bz]; v >= 0; v =
                blossom.next[v]) {
                if (dfs_augment(v, bz))
                    return true;
            }
        }
    }
    return false;
}

int find_maximal() {
    // discard blossoms whose potential is 0.
    for (int u = 1; u <= N; ++u)
        dsu.par[u] = u;
    for (int i = 0; i < dsu_changelog_size_; ++i) {
```

```
            dsu.par[dsu_changelog[i].v] = dsu_changelog[i].par;
        }
        for (int u = 1; u <= N; ++u) {
            label[u] = kFree;
            blossom.push(dsu.find(u), u);
        }
        int ret = 0;
        for (int u = 1; u <= N; ++u)
            if (!mate[u]) {
                int bu = dsu.par[u];
                if (label[bu] != kFree)
                    continue;
                label[bu] = outer_id_++;
                for (int v = blossom.head[bu]; v >= 0; v =
                     blossom.next[v]) {
                    if (!dfs_augment(v, bu))
                        continue;
                    ret += 1;
                    break;
                }
            }
        assert(ret >= 1);
        return ret;
    }

    // init

    void initialize() {
        que = Queue<int>(N);

        mate.assign(N + 1, 0);
        potential.assign(N + 1, 1);
        label.assign(N + 1, kFree);
        link.assign(N + 1, {0, 0});

        dsu_changelog.resize(N);

        dsu = DisjointSetUnion(N + 1);
        list = LinkedList(NH + 1, edges.size());

        blossom = LinkedList(N + 1, N + 1);
        stack.resize(N);
        stack_last_ = 0;
    }

   public:
    const int N, NH;
    vector<int> ofs;
    vector<Edge> edges;

    Queue<int> que;

    vector<int> mate, potential;
    vector<int> label;
    vector<Link> link;

    vector<Log> dsu_changelog;
    int dsu_changelog_last_, dsu_changelog_size_;

    DisjointSetUnion dsu;
    LinkedList list, blossom;
    vector<int> stack;
    int stack_last_;

    int time_current_, time_augment_;
    int contract_count_, outer_id_;
};

using Edge = MaximumMatching::Edge;

// usage example
int main() {
    int n = 3;
    vector<Edge> es = {{0, 1}, {1, 2}, {0, 1}};

    auto mm = MaximumMatching(n, es);
    auto ans = mm.maximum_matching();

    cout << ans << endl;
}
```

MaxGeneralWeightedMatchingEvlogv.cpp

Description: desc

04874e, 1080 lines

```
// Taken from https://gist.github.com/min-25/
     b984122f97dd7f72500e0bd6e49906ca

template <typename CostType, typename TotalCostType = int64_t>
class MaximumWeightedMatching {
    /*
      Maximum Weighted Matching in General Graphs.
      - O(nm \log(n)) time
      - O(n + m) space

      Note: each vertex is 1-indexed.

      Ref:
        Harold N. Gabow,
        "Data Structures for Weighted Matching and
         Extensions to b-matching and f-factors" (2016)
        (https://arxiv.org/abs/1611.07541)
    */
   public:
    using cost_t = CostType;
    using tcost_t = TotalCostType;

   private:
    enum Label { kSeparated = -2, kInner = -1, kFree = 0,
         kOuter = 1 };
    static constexpr cost_t Inf = cost_t(1) << (sizeof(cost_t)
         * 8 - 2);

   private:
    template <typename T>
    class BinaryHeap {
       public:
        struct Node {
            bool operator<(const Node &rhs) const { return
                 value < rhs.value; }

            T value;
            int id;
        };

        BinaryHeap() {}

        BinaryHeap(int N) : size_(0), node(N + 1), index(N, 0)
             {}

        int size() const { return size_; }

        bool empty() const { return size_ == 0; }

        void clear() {
            while (size_ > 0)
                index[node[size_--].id] = 0;
        }

        T min() const { return node[1].value; }

        int argmin() const { return node[1].id; }  // argmin ?
        T get_val(int id) const { return node[index[id]].value;
             }

        void pop() {
            if (size_ > 0)
                pop(1);
        }

        void erase(int id) {
            if (index[id])
                pop(index[id]);
        }

        bool has(int id) const { return index[id] != 0; }

        void update(int id, T v) {
            if (!has(id))
                return push(id, v);
            bool up = (v < node[index[id]].value);
            node[index[id]].value = v;
            if (up)
                up_heap(index[id]);
            else
                down_heap(index[id]);
        }

        void decrease_key(int id, T v) {
            if (!has(id))
                return push(id, v);
            if (v < node[index[id]].value)
                node[index[id]].value = v, up_heap(index[id]);
        }

        void push(int id, T v) {
            // assert(!has(id));
            index[id] = ++size_;
            node[size_] = {v, id};
            up_heap(size_);
        }

       private:
        void pop(int pos) {
            index[node[pos].id] = 0;
            if (pos == size_) {
                --size_;
                return;
            }
            bool up = (node[size_].value < node[pos].value);
            node[pos] = node[size_--];
            index[node[pos].id] = pos;
            if (up)
                up_heap(pos);
            else
                down_heap(pos);
        }

        void swap_node(int a, int b) {
            swap(node[a], node[b]);
            index[node[a].id] = a;
            index[node[b].id] = b;
        }

        void down_heap(int pos) {
            for (int k = pos, nk = k; 2 * k <= size_; k = nk) {
                if (node[2 * k] < node[nk])
                    nk = 2 * k;
```

```
            if (2 * k + 1 <= size_ && node[2 * k + 1] <
                 node[nk])
                nk = 2 * k + 1;
            if (nk == k)
                break;
            swap_node(k, nk);
        }
    }

    void up_heap(int pos) {
        for (int k = pos; k > 1 && node[k] < node[k >> 1];
             k >>= 1)
            swap_node(k, k >> 1);
    }

    int size_;
    vector<Node> node;
    vector<int> index;
};

template <typename Key>
class PairingHeaps {
   private:
    struct Node {
        Node() : prev(-1) {}  // "prev < 0" means the node
             is unused.
        Node(Key v) : key(v), child(0), next(0), prev(0) {}

        Key key;
        int child, next, prev;
    };

   public:
    PairingHeaps(int H, int N) : heap(H), node(N) {
        // It consists of 'H' Pairing heaps.
        // Each heap-node ID can appear at most 1 time(s)
             among heaps
        // and should be in [1, N).
    }

    void clear(int h) {
        if (heap[h])
            clear_rec(heap[h]), heap[h] = 0;
    }

    void clear_all() {
        for (size_t i = 0; i < heap.size(); ++i)
            heap[i] = 0;
        for (size_t i = 0; i < node.size(); ++i)
            node[i] = Node();
    }

    bool empty(int h) const { return !heap[h]; }

    bool used(int v) const { return node[v].prev >= 0; }

    Key min(int h) const { return node[heap[h]].key; }

    int argmin(int h) const { return heap[h]; }

    void pop(int h) {
        // assert(!empty(h));
        erase(h, heap[h]);
    }

    void push(int h, int v, Key key) {
        // assert(!used(v));
        node[v] = Node(key);
        heap[h] = merge(heap[h], v);
    }

    void erase(int h, int v) {
        if (!used(v))
            return;
        int w = two_pass_pairing(node[v].child);
        if (!node[v].prev)
            heap[h] = w;
        else {
            cut(v);
            heap[h] = merge(heap[h], w);
        }
        node[v].prev = -1;
    }

    void decrease_key(int h, int v, Key key) {
        if (!used(v))
            return push(h, v, key);
        if (!node[v].prev)
            node[v].key = key;
        else {
            cut(v);
            node[v].key = key;
            heap[h] = merge(heap[h], v);
        }
    }

   private:
    void clear_rec(int v) {
        for (; v; v = node[v].next) {
            if (node[v].child)
                clear_rec(node[v].child);
            node[v].prev = -1;
        }
    }

    inline void cut(int v) {
        auto &n = node[v];
        int pv = n.prev, nv = n.next;
        auto &pn = node[pv];
        if (pn.child == v)
            pn.child = nv;
        else
            pn.next = nv;
        node[nv].prev = pv;
        n.next = n.prev = 0;
    }

    int merge(int l, int r) {
        if (!l)
            return r;
        if (!r)
            return l;
        if (node[l].key > node[r].key)
            swap(l, r);
        int lc = node[r].next = node[l].child;
        node[l].child = node[lc].prev = r;
        return node[r].prev = l;
    }

    int two_pass_pairing(int root) {
        if (!root)
            return 0;
        int a = root;
        root = 0;
        while (a) {
            int b = node[a].next, na = 0;
            node[a].prev = node[a].next = 0;
            if (b)
                na = node[b].next, node[b].prev = node[b].
                     next = 0;
            a = merge(a, b);
            node[a].next = root;
            root = a;
            a = na;
        }
        int s = node[root].next;
        node[root].next = 0;
        while (s) {
            int t = node[s].next;
            node[s].next = 0;
            root = merge(root, s);
            s = t;
        }
        return root;
    }

   private:
    vector<int> heap;
    vector<Node> node;
};

template <typename T>
struct PriorityQueue : public priority_queue<T, vector<T>,
     greater<T> > {
    PriorityQueue() {}

    PriorityQueue(int N) { this->c.reserve(N); }

    T min() const { return this->top(); }

    void clear() { this->c.clear(); }
};

template <typename T>
struct Queue {
    Queue() {}

    Queue(int N) : qh(0), qt(0), data(N) {}

    T operator[](int i) const { return data[i]; }

    void enqueue(int u) { data[qt++] = u; }

    int dequeue() { return data[qh++]; }

    bool empty() const { return qh == qt; }

    void clear() { qh = qt = 0; }

    int size() const { return qt; }

    int qh, qt;
    vector<T> data;
};

public:
 struct InputEdge {
     int from, to;
     cost_t cost;
 };

private:
 template <typename T>
 using ModifiableHeap = BinaryHeap<T>;
 template <typename T>
 using ModifiableHeaps = PairingHeaps<T>;
 template <typename T>
```

```
  using FastHeap = PriorityQueue<T>;

  struct Edge {
      int to;
      cost_t cost;
  };
  struct Link {
      int from, to;
  };

  struct Node {
      struct NodeLink {
          int b, v;
      };

      Node() {}

      Node(int u) : parent(0), size(1) { link[0] = link[1] =
           {u, u}; }

      int next_v() const { return link[0].v; }

      int next_b() const { return link[0].b; }

      int prev_v() const { return link[1].v; }

      int prev_b() const { return link[1].b; }

      int parent, size;
      NodeLink link[2];
  };

  struct Event {
      Event() {}

      Event(cost_t time, int id) : time(time), id(id) {}

      bool operator<(const Event &rhs) const { return time <
           rhs.time; }

      bool operator>(const Event &rhs) const { return time >
           rhs.time; }

      cost_t time;
      int id;
  };

  struct EdgeEvent {
      EdgeEvent() {}

      EdgeEvent(cost_t time, int from, int to) : time(time),
           from(from), to(to) {}

      bool operator>(const EdgeEvent &rhs) const { return
           time > rhs.time; }

      bool operator<(const EdgeEvent &rhs) const { return
           time < rhs.time; }

      cost_t time;
      int from, to;
  };

 public:
  MaximumWeightedMatching(int N, const vector<InputEdge> &in)
      : N(N),
        B((N - 1) / 2),
        S(N + B + 1),
        ofs(N + 2),
        edges(in.size() * 2),
        heap2(S),
        heap2s(S, S),
        heap3(edges.size()),
        heap4(S) {
      for (auto &e : in)
          ofs[e.from + 1]++, ofs[e.to + 1]++;
      for (int i = 1; i <= N + 1; ++i)
          ofs[i] += ofs[i - 1];
      for (auto &e : in) {
          edges[ofs[e.from]++] = {e.to, e.cost * 2};
          edges[ofs[e.to]++] = {e.from, e.cost * 2};
      }
      for (int i = N + 1; i > 0; --i)
          ofs[i] = ofs[i - 1];
      ofs[0] = 0;
  }

  tcost_t maximum_weighted_matching(bool init_matching =
       false) {
      initialize();
      set_potential();
      if (init_matching)
          find_maximal_matching();
      for (int u = 1; u <= N; ++u)
          if (!mate[u])
              do_edmonds_search(u);
      tcost_t ret = compute_optimal_value();
      return ret;
  }

 private:
  tcost_t compute_optimal_value() const {
      tcost_t ret = 0;
      for (int u = 1; u <= N; ++u)
          if (mate[u] > u) {
              cost_t max_c = 0;
              for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid)
                   {
                  if (edges[eid].to == mate[u])
                      max_c = max(max_c, edges[eid].cost);
              }
              ret += max_c;
          }
      return ret >> 1;
  }

  inline tcost_t reduced_cost(int u, int v, const Edge &e)
       const {
      return tcost_t(potential[u]) + potential[v] - e.cost;
  }

  void rematch(int v, int w) {
      int t = mate[v];
      mate[v] = w;
      if (mate[t] != v)
          return;
      if (link[v].to == surface[link[v].to]) {
          mate[t] = link[v].from;
          rematch(mate[t], t);
      } else {
          int x = link[v].from, y = link[v].to;
          rematch(x, y);
          rematch(y, x);
      }
  }

  void fix_mate_and_base(int b) {
      if (b <= N)
          return;
      int bv = base[b], mv = node[bv].link[0].v, bmv = node[
           bv].link[0].b;
      int d = (node[bmv].link[1].v == mate[mv]) ? 0 : 1;
      while (1) {
          int mv = node[bv].link[d].v, bmv = node[bv].link[d
               ].b;
          if (node[bmv].link[1 ^ d].v != mate[mv])
              break;
          fix_mate_and_base(bv);
          fix_mate_and_base(bmv);
          bv = node[bmv].link[d].b;
      }
      fix_mate_and_base(base[b] = bv);
      mate[b] = mate[bv];
  }

  void reset_time() {
      time_current_ = 0;
      event1 = {Inf, 0};
  }

  void reset_blossom(int b) {
      label[b] = kFree;
      link[b].from = 0;
      slack[b] = Inf;
      lazy[b] = 0;
  }

  void reset_all() {
      label[0] = kFree;
      link[0].from = 0;
      for (int v = 1; v <= N; ++v) {  // should be optimized
           for sparse graphs.
          if (label[v] == kOuter)
              potential[v] -= time_current_;
          else {
              int bv = surface[v];
              potential[v] += lazy[bv];
              if (label[bv] == kInner)
                  potential[v] += time_current_ -
                       time_created[bv];
          }
          reset_blossom(v);
      }
      for (int b = N + 1, r = B - unused_bid_idx_; r > 0 && b
           < S; ++b)
          if (base[b] != b) {
              if (surface[b] == b) {
                  fix_mate_and_base(b);
                  if (label[b] == kOuter)
                      potential[b] += (time_current_ -
                           time_created[b]) << 1;
                  else if (label[b] == kInner)
                      fix_blossom_potential<kInner>(b);
                  else
                      fix_blossom_potential<kFree>(b);
              }
              heap2s.clear(b);
              reset_blossom(b);
              --r;
          }

      que.clear();
      reset_time();
      heap2.clear();
      heap3.clear();
      heap4.clear();
  }
```

```
void do_edmonds_search(int root) {
    if (potential[root] == 0)
        return;
    link_blossom(surface[root], {0, 0});
    push_outer_and_fix_potentials(surface[root], 0);
    for (bool augmented = false; !augmented;) {
        augmented = augment(root);
        if (augmented)
            break;
        augmented = adjust_dual_variables(root);
    }
    reset_all();
}

template <Label Lab>
inline cost_t fix_blossom_potential(int b) {
    // Return the amount.
    // (If v is an atom, the potential[v] will not be
         changed.)
    cost_t d = lazy[b];
    lazy[b] = 0;
    if (Lab == kInner) {
        cost_t dt = time_current_ - time_created[b];
        if (b > N)
            potential[b] -= dt << 1;
        d += dt;
    }
    return d;
}

template <Label Lab>
inline void update_heap2(int x, int y, int by, cost_t t) {
    if (t >= slack[y])
        return;
    slack[y] = t;
    best_from[y] = x;
    if (y == by) {
        if (Lab != kInner)
            heap2.decrease_key(y, EdgeEvent(t + lazy[y], x,
                y));
    } else {
        int gy = group[y];
        if (gy != y) {
            if (t >= slack[gy])
                return;
            slack[gy] = t;
        }
        heap2s.decrease_key(by, gy, EdgeEvent(t, x, y));
        if (Lab == kInner)
            return;
        EdgeEvent m = heap2s.min(by);
        heap2.decrease_key(by, EdgeEvent(m.time + lazy[by],
             m.from, m.to));
    }
}

void activate_heap2_node(int b) {
    if (b <= N) {
        if (slack[b] < Inf)
            heap2.push(b, EdgeEvent(slack[b] + lazy[b],
                best_from[b], b));
    } else {
        if (heap2s.empty(b))
            return;
        EdgeEvent m = heap2s.min(b);
        heap2.push(b, EdgeEvent(m.time + lazy[b], m.from, m
            .to));
    }
}

}

void swap_blossom(int a, int b) {
    // Assume that `b` is a maximal blossom.
    swap(base[a], base[b]);
    if (base[a] == a)
        base[a] = b;
    swap(heavy[a], heavy[b]);
    if (heavy[a] == a)
        heavy[a] = b;
    swap(link[a], link[b]);
    swap(mate[a], mate[b]);
    swap(potential[a], potential[b]);
    swap(lazy[a], lazy[b]);
    swap(time_created[a], time_created[b]);
    for (int d = 0; d < 2; ++d)
        node[node[a].link[d].b].link[1 ^ d].b = b;
    swap(node[a], node[b]);
}

void set_surface_and_group(int b, int sf, int g) {
    surface[b] = sf, group[b] = g;
    if (b <= N)
        return;
    for (int bb = base[b]; surface[bb] != sf; bb = node[bb
        ].next_b()) {
        set_surface_and_group(bb, sf, g);
    }
}

void merge_smaller_blossoms(int bid) {
    int lb = bid, largest_size = 1;
    for (int beta = base[bid], b = beta;;) {
        if (node[b].size > largest_size)
            largest_size = node[b].size, lb = b;
        if ((b = node[b].next_b()) == beta)
            break;
    }
    for (int beta = base[bid], b = beta;;) {
        if (b != lb)
            set_surface_and_group(b, lb, b);
        if ((b = node[b].next_b()) == beta)
            break;
    }
    group[lb] = lb;
    if (largest_size > 1) {
        surface[bid] = heavy[bid] = lb;
        swap_blossom(lb, bid);
    } else
        heavy[bid] = 0;
}

void contract(int x, int y, int eid) {
    int bx = surface[x], by = surface[y];
    assert(bx != by);
    const int h = -(eid + 1);
    link[surface[mate[bx]]].from = link[surface[mate[by]]].
        from = h;

    int lca = -1;
    while (1) {
        if (mate[by] != 0)
            swap(bx, by);
        bx = lca = surface[link[bx].from];
        if (link[surface[mate[bx]]].from == h)
            break;
        link[surface[mate[bx]]].from = h;
    }
    const int bid = unused_bid[--unused_bid_idx_];
    assert(unused_bid_idx_ >= 0);
    int tree_size = 0;
    for (int d = 0; d < 2; ++d) {
        for (int bv = surface[x]; bv != lca;) {
            int mv = mate[bv], bmv = surface[mv], v = mate[
                mv];
            int f = link[v].from, t = link[v].to;
            tree_size += node[bv].size + node[bmv].size;
            link[mv] = {x, y};

            if (bv > N)
                potential[bv] += (time_current_ -
                    time_created[bv]) << 1;
            if (bmv > N)
                heap4.erase(bmv);
            push_outer_and_fix_potentials(bmv,
                fix_blossom_potential<kInner>(bmv));

            node[bv].link[d] = {bmv, mv};
            node[bmv].link[1 ^ d] = {bv, v};
            node[bmv].link[d] = {bv = surface[f], f};
            node[bv].link[1 ^ d] = {bmv, t};
        }
        node[surface[x]].link[1 ^ d] = {surface[y], y};
        swap(x, y);
    }
    if (lca > N)
        potential[lca] += (time_current_ - time_created[lca
            ]) << 1;
    node[bid].size = tree_size + node[lca].size;
    base[bid] = lca;
    link[bid] = link[lca];
    mate[bid] = mate[lca];
    label[bid] = kOuter;
    surface[bid] = bid;
    time_created[bid] = time_current_;
    potential[bid] = 0;
    lazy[bid] = 0;

    merge_smaller_blossoms(bid);  // O(n log n) time /
        Edmonds search
}

void link_blossom(int v, Link l) {
    link[v] = {l.from, l.to};
    if (v <= N)
        return;
    int b = base[v];
    link_blossom(b, l);
    int pb = node[b].prev_b();
    l = {node[pb].next_v(), node[b].prev_v()};
    for (int bv = b;;) {
        int bw = node[bv].next_b();
        if (bw == b)
            break;
        link_blossom(bw, l);
        Link nl = {node[bw].prev_v(), node[bv].next_v()};
        bv = node[bw].next_b();
        link_blossom(bv, nl);
    }
}

void push_outer_and_fix_potentials(int v, cost_t d) {
    label[v] = kOuter;
    if (v > N) {
        for (int b = base[v]; label[b] != kOuter; b = node[
            b].next_b()) {
            push_outer_and_fix_potentials(b, d);
```

```
        }
    } else {
        potential[v] += time_current_ + d;
        if (potential[v] < event1.time)
            event1 = {potential[v], v};
        que.enqueue(v);
    }
}

bool grow(int x, int y) {
    int by = surface[y];
    bool visited = (label[by] != kFree);
    if (!visited)
        link_blossom(by, {0, 0});
    label[by] = kInner;
    time_created[by] = time_current_;
    heap2.erase(by);
    if (y != by)
        heap4.update(by, time_current_ + (potential[by] >>
            1));
    int z = mate[by];
    if (z == 0) {
        rematch(x, y);
        rematch(y, x);
        return true;
    }
    int bz = surface[z];
    if (!visited)
        link_blossom(bz, {x, y});
    else
        link[bz] = link[z] = {x, y};
    push_outer_and_fix_potentials(bz, fix_blossom_potential
        <kFree>(bz));
    time_created[bz] = time_current_;
    heap2.erase(bz);
    return false;
}

void free_blossom(int bid) {
    unused_bid[unused_bid_idx_++] = bid;
    base[bid] = bid;
}

int recalculate_minimum_slack(int b, int g) {
    // Return the destination of the best edge of blossom `
        g`.
    if (b <= N) {
        if (slack[b] >= slack[g])
            return 0;
        slack[g] = slack[b];
        best_from[g] = best_from[b];
        return b;
    }
    int v = 0;
    for (int beta = base[b], bb = beta;;) {
        int w = recalculate_minimum_slack(bb, g);
        if (w != 0)
            v = w;
        if ((bb = node[bb].next_b()) == beta)
            break;
    }
    return v;
}

void construct_smaller_components(int b, int sf, int g) {
    surface[b] = sf, group[b] = g;  // `group[b] = g` is
        unneeded.
    if (b <= N)
        return;
    for (int bb = base[b]; surface[bb] != sf; bb = node[bb
        ].next_b()) {
        if (bb == heavy[b]) {
            construct_smaller_components(bb, sf, g);
        } else {
            set_surface_and_group(bb, sf, bb);
            int to = 0;
            if (bb > N)
                slack[bb] = Inf, to =
                    recalculate_minimum_slack(bb, bb);
            else if (slack[bb] < Inf)
                to = bb;
            if (to > 0)
                heap2s.push(sf, bb, EdgeEvent(slack[bb],
                    best_from[bb], to));
        }
    }
}

void move_to_largest_blossom(int bid) {
    const int h = heavy[bid];
    cost_t d = (time_current_ - time_created[bid]) + lazy[
        bid];
    lazy[bid] = 0;
    for (int beta = base[bid], b = beta;;) {
        time_created[b] = time_current_;
        lazy[b] = d;
        if (b != h)
            construct_smaller_components(b, b, b), heap2s.
                erase(bid, b);
        if ((b = node[b].next_b()) == beta)
            break;
    }
    if (h > 0)
        swap_blossom(h, bid), bid = h;
    free_blossom(bid);
}

void expand(int bid) {
    int mv = mate[base[bid]];
    move_to_largest_blossom(bid);  // O(n log n) time /
        Edmonds search
    Link old_link = link[mv];
    int old_base = surface[mate[mv]], root = surface[
        old_link.to];
    int d = (mate[root] == node[root].link[0].v) ? 1 : 0;
    for (int b = node[old_base].link[d ^ 1].b; b != root;)
        {
        label[b] = kSeparated;
        activate_heap2_node(b);
        b = node[b].link[d ^ 1].b;
        label[b] = kSeparated;
        activate_heap2_node(b);
        b = node[b].link[d ^ 1].b;
    }
    for (int b = old_base;; b = node[b].link[d].b) {
        label[b] = kInner;
        int nb = node[b].link[d].b;
        if (b == root)
            link[mate[b]] = old_link;
        else
            link[mate[b]] = {node[b].link[d].v, node[nb].
                link[d ^ 1].v};
        link[surface[mate[b]]] = link[mate[b]];  // fix
            tree links
        if (b > N) {
            if (potential[b] == 0)
                expand(b);
            else
                heap4.push(b, time_current_ + (potential[b]
                    >> 1));
        }
        if (b == root)
            break;
        push_outer_and_fix_potentials(nb,
            fix_blossom_potential<kInner>(b = nb));
    }
}

bool augment(int root) {
    // Return true if an augmenting path is found.
    while (!que.empty()) {
        int x = que.dequeue(), bx = surface[x];
        if (potential[x] == time_current_) {
            if (x != root)
                rematch(x, 0);
            return true;
        }
        for (int eid = ofs[x]; eid < ofs[x + 1]; ++eid) {
            auto &e = edges[eid];
            int y = e.to, by = surface[y];
            if (bx == by)
                continue;
            Label l = label[by];
            if (l == kOuter) {
                cost_t t = reduced_cost(x, y, e) >> 1;  //
                    < 2 * Inf
                if (t == time_current_) {
                    contract(x, y, eid);
                    bx = surface[x];
                } else if (t < event1.time) {
                    heap3.emplace(t, x, eid);
                }
            } else {
                tcost_t t = reduced_cost(x, y, e);  // < 3
                    * Inf
                if (t >= Inf)
                    continue;
                if (l != kInner) {
                    if (cost_t(t) + lazy[by] ==
                        time_current_) {
                        if (grow(x, y))
                            return true;
                    } else
                        update_heap2<kFree>(x, y, by, t);
                } else {
                    if (mate[x] != y)
                        update_heap2<kInner>(x, y, by, t);
                }
            }
        }
    }
    return false;
}

bool adjust_dual_variables(int root) {
    // delta1 : rematch
    cost_t time1 = event1.time;

    // delta2 : grow
    cost_t time2 = Inf;
    if (!heap2.empty())
        time2 = heap2.min().time;

    // delta3 : contract : O(m log n) time / Edmonds search
        [ bottleneck (?) ]
    cost_t time3 = Inf;
    while (!heap3.empty()) {
```

```
            EdgeEvent e = heap3.min();
            int x = e.from, y = edges[e.to].to;  // e.to is some edge id.
            if (surface[x] != surface[y]) {
                time3 = e.time;
                break;
            } else
                heap3.pop();
        }

        // delta4 : expand
        cost_t time4 = Inf;
        if (!heap4.empty())
            time4 = heap4.min();

        // -- events --
        cost_t time_next = min(min(time1, time2), min(time3, time4));
        assert(time_current_ <= time_next && time_next < Inf);
        time_current_ = time_next;

        if (time_current_ == event1.time) {
            int x = event1.id;
            if (x != root)
                rematch(x, 0);
            return true;
        }
        while (!heap2.empty() && heap2.min().time == time_current_) {
            int x = heap2.min().from, y = heap2.min().to;
            if (grow(x, y))
                return true;  // `grow` function will call `heap2.erase(by)`.
        }
        while (!heap3.empty() && heap3.min().time == time_current_) {
            int x = heap3.min().from, eid = heap3.min().to;
            int y = edges[eid].to;
            heap3.pop();
            if (surface[x] == surface[y])
                continue;
            contract(x, y, eid);
        }
        while (!heap4.empty() && heap4.min() == time_current_) {
            int b = heap4.argmin();
            heap4.pop();
            expand(b);
        }
        return false;
    }

private:
    void initialize() {
        que = Queue<int>(N);
        mate.assign(S, 0);
        link.assign(S, {0, 0});
        label.assign(S, kFree);
        base.resize(S);
        for (int u = 1; u < S; ++u)
            base[u] = u;
        surface.resize(S);
        for (int u = 1; u < S; ++u)
            surface[u] = u;

        potential.resize(S);
        node.resize(S);
        for (int b = 1; b < S; ++b)
            node[b] = Node(b);
        unused_bid.resize(B);
        for (int i = 0; i < B; ++i)
            unused_bid[i] = N + B - i;
        unused_bid_idx_ = B;

        // for O(nm log n) implementation
        reset_time();
        time_created.resize(S);
        slack.resize(S);
        for (int i = 0; i < S; ++i)
            slack[i] = Inf;
        best_from.assign(S, 0);
        heavy.assign(S, 0);
        lazy.assign(S, 0);
        group.resize(S);
        for (int i = 0; i < S; ++i)
            group[i] = i;
    }

    void set_potential() {
        for (int u = 1; u <= N; ++u) {
            cost_t max_c = 0;
            for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
                max_c = max(max_c, edges[eid].cost);
            }
            potential[u] = max_c >> 1;
        }
    }

    void find_maximal_matching() {
        // Find a maximal matching naively.
        for (int u = 1; u <= N; ++u)
            if (!mate[u]) {
                for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
                    auto &e = edges[eid];
                    int v = e.to;
                    if (mate[v] > 0 || reduced_cost(u, v, e) > 0)
                        continue;
                    mate[u] = v;
                    mate[v] = u;
                    break;
                }
            }
    }

private:
    const int N, B, S;  // N = |V|, B = (|V| - 1) / 2, S = N + B + 1
    vector<int> ofs;
    vector<Edge> edges;

    Queue<int> que;
    vector<int> mate, surface, base;
    vector<Link> link;
    vector<Label> label;
    vector<cost_t> potential;

    vector<int> unused_bid;
    int unused_bid_idx_;
    vector<Node> node;

    // for O(nm log n) implementation
    vector<int> heavy, group;
    vector<cost_t> time_created, lazy, slack;
    vector<int> best_from;
    cost_t time_current_;
    Event event1;
    ModifiableHeap<EdgeEvent> heap2;
    ModifiableHeaps<EdgeEvent> heap2s;
    FastHeap<EdgeEvent> heap3;
    ModifiableHeap<cost_t> heap4;
};

using MWM = MaximumWeightedMatching<int>;
using Edge = MWM::InputEdge;

// ----

int main() {
    const int N = 3000, B = 1, C = 5e8;
    uniform_int_distribution<int> urand(B, C);
    mt19937 mt(12345);

    auto run = [](int N, const vector<Edge> &edges) {
        clock_t beg = clock();
        auto mwm = MWM(N, edges);
        auto ans = mwm.maximum_weighted_matching();
        clock_t end = clock();
        fprintf(stderr, "|V| = %6d, |E| = %7d: %lld %.3f sec\n"
            , N, (int)edges.size(), ans,
                double(end - beg) / CLOCKS_PER_SEC);
    };

    // Complete
    {
        vector<Edge> edges;
        for (int i = 0; i < N; ++i) {
            for (int j = i + 1; j < N; ++j) {
                edges.push_back({i + 1, j + 1, urand(mt)});
            }
        }
        run(N, edges);
    }
    // Complete Bipartite
    {
        vector<Edge> edges;
        for (int i = 0; i < N / 2; ++i) {
            for (int j = 0; j < N - N / 2; ++j) {
                edges.push_back({i + 1, N / 2 + j + 1, urand(mt)});
            }
        }
        run(N, edges);
    }
    // Random Graph
    {
        const int N2 = N * 3;
        const int M = N2 * 30;
        vector<Edge> edges(M);
        uniform_int_distribution<int> gene(1, N2);
        for (int i = 0; i < M; ++i) {
            int u = gene(mt), v = gene(mt), c = urand(mt);
            edges[i] = {u, v, c};
        }
        run(N2, edges);
    }
}
```

MaxGeneralWeightedMatchingV3.cpp

Description: desc

7b9923, 281 lines

```
// Taken from http://uoj.ac/submission/187480

// N^3 (but fast in practice)
```

```cpp
const int INF = INT_MAX;
const int N = 514;

struct edge {
    int u, v, w;
};

int n, n_x;
edge g[N * 2][N * 2];
int lab[N * 2];
int match[N * 2], slack[N * 2], st[N * 2], pa[N * 2];
int flower_from[N * 2][N + 1], S[N * 2], vis[N * 2];
vector<int> flower[N * 2];
queue<int> q;

int e_delta(const edge &e) {
    return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
}

void update_slack(int u, int x) {
    if (!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x])
        )
        slack[x] = u;
}

void set_slack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; ++u)
        if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
            update_slack(u, x);
}

void q_push(int x) {
    if (x <= n)
        q.push(x);
    else
        for (int i : flower[x])
            q_push(i);
}

void set_st(int x, int b) {
    st[x] = b;
    if (x > n)
        for (int i : flower[x])
            set_st(i, b);
}

int get_pr(int b, int xr) {
    int pr = find(flower[b].begin(), flower[b].end(), xr) -
        flower[b].begin();
    if (pr % 2 == 1) {
        reverse(flower[b].begin() + 1, flower[b].end());
        return (int)flower[b].size() - pr;
    } else
        return pr;
}

void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n)
        return;
    edge e = g[u][v];
    int xr = flower_from[u][e.u], pr = get_pr(u, xr);
    for (int i = 0; i < pr; ++i)
        set_match(flower[u][i], flower[u][i ^ 1]);
    set_match(xr, v);
    rotate(flower[u].begin(), flower[u].begin() + pr, flower[u
        ].end());
}

void augment(int u, int v) {
    for (;;) {
        int xnv = st[match[u]];
        set_match(u, v);
        if (!xnv)
            return;
        set_match(xnv, st[pa[xnv]]);
        u = st[pa[xnv]], v = xnv;
    }
}

int get_lca(int u, int v) {
    static int t = 0;
    for (++t; u || v; swap(u, v)) {
        if (u == 0)
            continue;
        if (vis[u] == t)
            return u;
        vis[u] = t;
        u = st[match[u]];
        if (u)
            u = st[pa[u]];
    }
    return 0;
}

void add_blossom(int u, int lca, int v) {
    int b = n + 1;
    while (b <= n_x && st[b])
        ++b;
    if (b > n_x)
        ++n_x;
    lab[b] = 0, S[b] = 0;
    match[b] = match[lca];
    flower[b].clear();
    flower[b].push_back(lca);
    for (int x = u, y; x != lca; x = st[pa[y]])
        flower[b].push_back(x), flower[b].push_back(y = st[
            match[x]]), q_push(y);
    reverse(flower[b].begin() + 1, flower[b].end());
    for (int x = v, y; x != lca; x = st[pa[y]])
        flower[b].push_back(x), flower[b].push_back(y = st[
            match[x]]), q_push(y);
    set_st(b, b);
    for (int x = 1; x <= n_x; ++x)
        g[b][x].w = g[x][b].w = 0;
    for (int x = 1; x <= n; ++x)
        flower_from[b][x] = 0;
    for (size_t i = 0; i < flower[b].size(); ++i) {
        int xs = flower[b][i];
        for (int x = 1; x <= n_x; ++x)
            if (g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g
                [b][x]))
                g[b][x] = g[xs][x], g[x][b] = g[x][xs];
        for (int x = 1; x <= n; ++x)
            if (flower_from[xs][x])
                flower_from[b][x] = xs;
    }
    set_slack(b);
}

void expand_blossom(int b) {
    for (int i : flower[b])
        set_st(i, i);
    int xr = flower_from[b][g[b][pa[b]].u];
    int pr = get_pr(b, xr);
    for (int i = 0; i < pr; i += 2) {
        int xs = flower[b][i], xns = flower[b][i + 1];
        pa[xs] = g[xns][xs].u;
        S[xs] = 1;
        S[xns] = 0;
        slack[xs] = 0;
        set_slack(xns);
        q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for (int i = pr + 1; i < flower[b].size(); ++i) {
        int xs = flower[b][i];
        S[xs] = -1;
        set_slack(xs);
    }
    st[b] = 0;
}

bool on_found_edge(const edge &e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) {
        pa[v] = e.u;
        S[v] = 1;
        int nu = st[match[v]];
        slack[v] = slack[nu] = 0;
        S[nu] = 0;
        q_push(nu);
    } else if (S[v] == 0) {
        int lca = get_lca(u, v);
        if (!lca) {
            augment(u, v);
            augment(v, u);
            return true;
        } else {
            add_blossom(u, lca, v);
        }
    }
    return false;
}

bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && !match[x]) {
            pa[x] = 0;
            S[x] = 0;
            q_push(x);
        }
    if (q.empty())
        return false;
    for (;;) {
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            if (S[st[u]] == 1)
                continue;
            for (int v = 1; v <= n; ++v)
                if (g[u][v].w > 0 && st[u] != st[v]) {
                    if (e_delta(g[u][v]) == 0) {
                        if (on_found_edge(g[u][v]))
                            return true;
                    } else
                        update_slack(u, st[v]);
                }
        }
        int d = INF;
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b && S[b] == 1)
                d = min(d, lab[b] / 2);
```

```
        for (int x = 1; x <= n_x; ++x)
            if (st[x] == x && slack[x]) {
                if (S[x] == -1)
                    d = min(d, e_delta(g[slack[x]][x]));
                else if (S[x] == 0)
                    d = min(d, e_delta(g[slack[x]][x]) / 2);
            }
        for (int u = 1; u <= n; ++u) {
            if (S[st[u]] == 0) {
                if (lab[u] <= d)
                    return false;
                lab[u] -= d;
            } else if (S[st[u]] == 1)
                lab[u] += d;
        }
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b) {
                if (S[st[b]] == 0)
                    lab[b] += d * 2;
                else if (S[st[b]] == 1)
                    lab[b] -= d * 2;
            }
        q = queue<int>();
        for (int x = 1; x <= n_x; ++x)
            if (st[x] == x && slack[x] && st[slack[x]] != x &&
                e_delta(g[slack[x]][x]) == 0)
                if (on_found_edge(g[slack[x]][x]))
                    return true;
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b && S[b] == 1 && lab[b] == 0)
                expand_blossom(b);
    }
}

tuple<long long, int> solve() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int u = 0; u <= n; ++u) {
        st[u] = u;
        flower[u].clear();
    }
    int w_max = 0;
    for (int u = 1; u <= n; ++u)
        for (int v = 1; v <= n; ++v) {
            flower_from[u][v] = (u == v ? u : 0);
            w_max = max(w_max, g[u][v].w);
        }
    for (int u = 1; u <= n; ++u)
        lab[u] = w_max;
    while (matching())
        ++n_matches;
    for (int u = 1; u <= n; ++u)
        if (match[u] && match[u] < u)
            tot_weight += g[u][match[u]].w;
    return {tot_weight, n_matches};
}

void add_edge(int u, int v, int w) {
    g[u][v].w = g[v][u].w = w;
}

void init(int _n) {
    n = _n;
    for (int u = 1; u <= n; ++u)
        for (int v = 1; v <= n; ++v)
            g[u][v] = edge{u, v, 0};
}
```

```
// usage example
int main() {
    int n = 4;
    init(n);
    add_edge(1, 2, 4);
    auto [tot_weight, n_matches] = solve();
    cout << tot_weight << " " << n_matches << endl;
}
```

MinBipartiteWeightedMatchingHungarian.cpp

Description: desc

18b15d, 57 lines

```
// https://en.wikipedia.org/wiki/Hungarian_algorithm in O(n^2 *
    m)
pair<int, vector<int>> min_weight_perfect_matching(const vector
    <vector<int>> &a) {
    int n = a.size();
    int m = a[0].size();
    vector<int> u(n);
    vector<int> v(m);
    vector<int> p(m);
    vector<int> way(m);
    for (int i = 1; i < n; ++i) {
        vector<int> minv(m, numeric_limits<int>::max());
        vector<bool> used(m);
        p[0] = i;
        int j0 = 0;
        while (p[j0] != 0) {
            used[j0] = true;
            int i0 = p[j0];
            int delta = numeric_limits<int>::max();
            int j1 = 0;
            for (int j = 1; j < m; ++j)
                if (!used[j]) {
                    int d = a[i0][j] - u[i0] - v[j];
                    if (minv[j] > d) {
                        minv[j] = d;
                        way[j] = j0;
                    }
                    if (delta > minv[j]) {
                        delta = minv[j];
                        j1 = j;
                    }
                }
            for (int j = 0; j < m; ++j)
                if (used[j]) {
                    u[p[j]] += delta;
                    v[j] -= delta;
                } else
                    minv[j] -= delta;
            j0 = j1;
        }
        while (j0 != 0) {
            int j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        }
    }
    vector<int> matching(n);
    for (int i = 1; i < m; ++i)
        matching[p[i]] = i;
    return {-v[0], matching};
}

// usage example
int main() {
    // row1 and col1 should contain 0
    vector<vector<int>> a{{0, 0, 0}, {0, 1, 2}, {0, 1, 2}};
    auto [cost, matching] = min_weight_perfect_matching(a);
    cout << boolalpha << (cost == 3) << endl;
}
```

MinCostCirculation.cpp

Description: desc

53df05, 91 lines

```
// Negative cycles canceling algorithm in O(E^2 * V *
    MAX_EDGE_FLOW * MAX_EDGE_COST)
// negative-cost edges are allowed
// negative-cost cycles are allowed

struct Edge {
    int to, rev, f, cap, cost;
};

struct min_cost_circulation {
    vector<vector<Edge>> graph;

    min_cost_circulation(int nodes) : graph(nodes) {}

    void add_edge(int s, int t, int cap, int cost) {
        Edge a = {t, (int)graph[t].size(), 0, cap, cost};
        Edge b = {s, (int)graph[s].size(), 0, 0, -cost};
        graph[s].emplace_back(a);
        graph[t].emplace_back(b);
    }

    int calc_min_cost_circulation() {
        int circulation_cost = 0;
        int n = graph.size();
        constexpr int INF = 1000'000'000;
        for (bool found = true; found;) {
            found = false;
            vector<int> dist(n, INF);
            vector<int> p(n, -1);
            for (int i = 0; i < n; ++i) {
                if (dist[i] == INF) {
                    dist[i] = 0;
                    vector<int> q;
                    q.emplace_back(i);
                    for (int j = 0; j < n && !q.empty(); ++j) {
                        vector<int> nq;
                        for (int u : q)
                            for (auto &e : graph[u])
                                if (e.f < e.cap)
                                    if (dist[e.to] > dist[u] +
                                        e.cost) {
                                        dist[e.to] = dist[u] +
                                            e.cost;
                                        p[e.to] = u;
                                        nq.push_back(e.to);
                                    }
                        q = nq;
                        sort(q.begin(), q.end());
                        q.erase(unique(q.begin(), q.end()), q.
                            end());
                    }
                    if (!q.empty()) {
                        int leaf = q[0];
                        vector<int> path;
                        for (int u = leaf; u != -1; u = p[u])
                            if (find(path.begin(), path.end(),
                                u) == path.end())
                                path.push_back(u);
                            else {
```

```
                            path.erase(path.begin(), find(
                                path.begin(), path.end(),
                                u));
                            break;
                        }
                    for (size_t j = 0; j < path.size(); ++j
                        ) {
                        int to = path[j];
                        int u = path[(j + 1) % path.size()
                            ];
                        for (auto &e : graph[u]) {
                            if (e.to == to) {
                                e.f += 1;
                                graph[u][e.rev].f -= 1;
                                circulation_cost += e.cost;
                            }
                        }
                    }
                    found = true;
                }
            }
        }
        return circulation_cost;
    }
};

// Usage example
int main() {
    int capacity[][3] = {{0, 1, 0}, {0, 0, 1}, {1, 0, 0}};
    int cost[][3] = {{0, -1, 0}, {0, 0, 1}, {-1, 0, 0}};
    int nodes = 3;
    min_cost_circulation mcf(nodes);
    for (int i = 0; i < nodes; i++)
        for (int j = 0; j < nodes; j++)
            if (capacity[i][j] != 0)
                mcf.add_edge(i, j, capacity[i][j], cost[i][j]);

    int circulation_cost = mcf.calc_min_cost_circulation();
    cout << circulation_cost << endl;
}
```

MinCostFlowBf.cpp

Description: desc

ddfd25, 91 lines

```
// https://en.wikipedia.org/wiki/Minimum-cost_flow_problem in O
    (E * V * FLOW)
// negative-cost edges are allowed
// negative-cost cycles are not allowed

struct Edge {
    int to, rev, cap, f, cost;
};

struct min_cost_flow {
    vector<vector<Edge>> graph;

    min_cost_flow(int nodes) : graph(nodes) {}

    void add_edge(int s, int t, int cap, int cost) {
        Edge a = {t, (int)graph[t].size(), cap, 0, cost};
        Edge b = {s, (int)graph[s].size(), 0, 0, -cost};
        graph[s].emplace_back(a);
        graph[t].emplace_back(b);
    }

    void bellman_ford(int s, vector<int> &curflow, vector<int>
         &dist, vector<int> &prevnode, vector<int> &prevedge) {
        int n = graph.size();
        vector<int> q(n);
        vector<bool> inqueue(n);
        fill(dist.begin(), dist.end(), numeric_limits<int>::max
            ());
        dist[s] = 0;
        curflow[s] = numeric_limits<int>::max();
        int qt = 0;
        q[qt++] = s;
        for (int qh = 0; qh != qt; qh++) {
            int u = q[qh % n];
            inqueue[u] = false;
            for (size_t i = 0; i < graph[u].size(); i++) {
                Edge &e = graph[u][i];
                if (e.cap <= e.f)
                    continue;
                int v = e.to;
                int ndist = dist[u] + e.cost;
                if (dist[v] > ndist) {
                    dist[v] = ndist;
                    prevnode[v] = u;
                    prevedge[v] = i;
                    curflow[v] = min(curflow[u], e.cap - e.f);
                    if (!inqueue[v]) {
                        inqueue[v] = true;
                        q[qt++ % n] = v;
                    }
                }
            }
        }
    }

    tuple<int, int> calc_min_cost_flow(int s, int t, int maxf)
        {
        int flow = 0;
        int flow_cost = 0;
        size_t n = graph.size();
        vector<int> curflow(n), dist(n), prevnode(n), prevedge(
            n);
        while (flow < maxf) {
            bellman_ford(s, curflow, dist, prevnode, prevedge);
            if (dist[t] == numeric_limits<int>::max())
                break;
            int df = min(curflow[t], maxf - flow);
            flow += df;
            for (int v = t; v != s; v = prevnode[v]) {
                Edge &e = graph[prevnode[v]][prevedge[v]];
                e.f += df;
                graph[v][e.rev].f -= df;
                flow_cost += df * e.cost;
            }
        }
        return {flow, flow_cost};
    }
};

// Usage example
int main() {
    int capacity[][3] = {{0, 3, 2}, {0, 0, 2}, {0, 0, 0}};
    int nodes = 3;
    min_cost_flow mcf(nodes);
    for (int i = 0; i < nodes; i++)
        for (int j = 0; j < nodes; j++)
            if (capacity[i][j] != 0)
                mcf.add_edge(i, j, capacity[i][j], 1);

    int s = 0;
    int t = 2;
    auto [flow, flow_cost] = mcf.calc_min_cost_flow(s, t,
        numeric_limits<int>::max());
    cout << (4 == flow) << endl;
    cout << (6 == flow_cost) << endl;
}
```

MinCostFlowDijkstra.cpp

Description: desc

"../../structures/binary_heap_indexed.h" 02a224, 123 lines

```
// https://cp-algorithms.com/graph/min_cost_flow.html in O(E *
    V + E * logV * FLOW)
// negative-cost edges are allowed
// negative-cost cycles are not allowed

struct Edge {
    int to, rev, cap, f, cost;
};

struct min_cost_flow {
    vector<vector<Edge>> graph;

    min_cost_flow(int nodes) : graph(nodes) {}

    void add_edge(int s, int t, int cap, int cost) {
        Edge a = {t, (int)graph[t].size(), cap, 0, cost};
        Edge b = {s, (int)graph[s].size(), 0, 0, -cost};
        graph[s].emplace_back(a);
        graph[t].emplace_back(b);
    }

    void bellman_ford(int s, vector<int> &dist) {
        int n = graph.size();
        vector<int> q(n);
        vector<bool> inqueue(n);
        fill(dist.begin(), dist.end(), numeric_limits<int>::max
            ());
        dist[s] = 0;
        int qt = 0;
        q[qt++] = s;
        for (int qh = 0; qh != qt; qh++) {
            int u = q[qh % n];
            inqueue[u] = false;
            for (auto &e : graph[u]) {
                if (e.cap <= e.f)
                    continue;
                int v = e.to;
                int ndist = dist[u] + e.cost;
                if (dist[v] > ndist) {
                    dist[v] = ndist;
                    if (!inqueue[v]) {
                        inqueue[v] = true;
                        q[qt++ % n] = v;
                    }
                }
            }
        }
    }

    void dijkstra(int s, int t, vector<int> &pot, vector<int> &
        dist, vector<bool> &finished, vector<int> &curflow,
            vector<int> &prevnode, vector<int> &prevedge)
                {
        binary_heap_indexed<int> h(graph.size());
        h.add(s, 0);
        fill(dist.begin(), dist.end(), numeric_limits<int>::max
            ());
        dist[s] = 0;
        curflow[s] = numeric_limits<int>::max();
        fill(finished.begin(), finished.end(), false);
        while (!finished[t] && h.size != 0) {
```

```
            int u = h.remove_min();
            finished[u] = true;
            for (size_t i = 0; i < graph[u].size(); i++) {
                Edge &e = graph[u][i];
                if (e.f >= e.cap)
                    continue;
                int v = e.to;
                int nprio = dist[u] + e.cost + pot[u] - pot[v];

                if (dist[v] > nprio) {
                    if (dist[v] == numeric_limits<int>::max())
                        h.add(v, nprio);
                    else
                        h.change_value(v, nprio);
                    dist[v] = nprio;
                    prevnode[v] = u;
                    prevedge[v] = i;
                    curflow[v] = min(curflow[u], e.cap - e.f);
                }
            }
        }
    }

    tuple<int, int> cal_min_cost_flow(int s, int t, int maxf) {
        size_t n = graph.size();
        vector<int> pot(n), curflow(n), dist(n), prevnode(n),
             prevedge(n);
        vector<bool> finished(n);
        bellman_ford(s, pot);  // this can be commented out if
             edges costs are non-negative
        int flow = 0;
        int flow_cost = 0;
        while (flow < maxf) {
            dijkstra(s, t, pot, dist, finished, curflow,
                 prevnode, prevedge);
            if (dist[t] == numeric_limits<int>::max())
                break;
            for (size_t i = 0; i < n; i++)
                if (finished[i])
                    pot[i] += dist[i] - dist[t];
            int df = min(curflow[t], maxf - flow);
            flow += df;
            for (int v = t; v != s; v = prevnode[v]) {
                Edge &e = graph[prevnode[v]][prevedge[v]];
                e.f += df;
                graph[v][e.rev].f -= df;
                flow_cost += df * e.cost;
            }
        }
        return {flow, flow_cost};
    }
};

// Usage example
int main() {
    int capacity[][3] = {{0, 3, 2}, {0, 0, 2}, {0, 0, 0}};
    int nodes = 3;
    min_cost_flow mcf(nodes);
    for (int i = 0; i < nodes; i++)
        for (int j = 0; j < nodes; j++)
            if (capacity[i][j] != 0)
                mcf.add_edge(i, j, capacity[i][j], 1);

    int s = 0;
    int t = 2;
    auto [flow, flow_cost] = mcf.cal_min_cost_flow(s, t,
         numeric_limits<int>::max());

    cout << (4 == flow) << endl;
    cout << (6 == flow_cost) << endl;
}
```

PrimHeap.cpp

Description: desc

8d29d7, 40 lines

```
using edge = pair<int, int>;  // (v, cost)
using item = pair<int, int>;  // (cost, v)

// Prim's algorithm in O(E*log(V)) time: https://cp-algorithms.
     com/graph/mst_prim.html
tuple<long long, vector<int>> prim_mst(const vector<vector<edge
     >> &g) {
    size_t n = g.size();
    vector<int> pred(n, -1);
    vector<int> prio(n, numeric_limits<int>::max());
    priority_queue<item, vector<item>, greater<>> q;
    q.emplace(prio[0] = 0, 0);
    long long tree_cost = 0;

    while (!q.empty()) {
        auto [c, u] = q.top();
        q.pop();

        if (prio[u] != c)
            continue;
        tree_cost += c;

        for (auto [v, cost] : g[u]) {
            if (prio[v] > cost) {
                prio[v] = cost;
                pred[v] = u;
                q.emplace(cost, v);
            }
        }
    }
    return {tree_cost, pred};
}

// usage example
int main() {
    vector<vector<edge>> graph = {{edge{1, 10}, edge{2, 5}}, {
         edge{0, 10}, edge{2, 10}}, {edge{1, 10}, edge{0, 5}}};
    auto [tree_weight, pred] = prim_mst(graph);
    cout << tree_weight << endl;
    for (int x : pred)
        cout << x << " ";
    cout << endl;
}
```

StableMatching.cpp

Description: desc

6538f6, 34 lines

```
vector<int> stable_matching(vector<vector<int>> prefer_m,
     vector<vector<int>> prefer_w) {
    int n = prefer_m.size();
    vector<int> pair_m(n, -1);
    vector<int> pair_w(n, -1);
    vector<int> p(n);
    for (int i = 0; i < n; i++) {
        while (pair_m[i] < 0) {
            int w = prefer_m[i][p[i]++];
            int m = pair_w[w];
            if (m == -1) {
                pair_m[i] = w;
                pair_w[w] = i;
            } else if (prefer_w[w][i] < prefer_w[w][m]) {
                pair_m[m] = -1;
                pair_m[i] = w;
                pair_w[w] = i;
                i = m;
            }
        }
    }
    return pair_m;
}

// usage example
int main() {
    vector<vector<int>> prefer_m{{0, 1, 2}, {0, 2, 1}, {1, 0, 2
         }};

    vector<vector<int>> prefer_w{{0, 1, 2}, {2, 0, 1}, {2, 1, 0
         }};

    vector<int> matching = stable_matching(prefer_m, prefer_w);
    for (int x : matching)
        cout << x << " ";
    cout << endl;
}
```

StronglyConnectedComponents.cpp

Description: desc

bf930c, 71 lines

```
// https://en.wikipedia.org/wiki/Kosaraju%27s_algorithm

void dfs(const vector<vector<int>> &graph, vector<bool> &used,
     vector<int> &res, int u) {
    used[u] = true;
    for (int v : graph[u])
        if (!used[v])
            dfs(graph, used, res, v);
    res.push_back(u);
}

// Get strongly connected components
vector<vector<int>> scc(const vector<vector<int>> &graph) {
    int n = graph.size();
    vector<bool> used(n);
    vector<int> order;
    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs(graph, used, order, i);

    vector<vector<int>> reverseGraph(n);
    for (int i = 0; i < n; i++)
        for (int j : graph[i])
            reverseGraph[j].push_back(i);

    vector<vector<int>> components;
    fill(used.begin(), used.end(), false);
    reverse(order.begin(), order.end());

    for (int u : order)
        if (!used[u]) {
            vector<int> component;
            dfs(reverseGraph, used, component, u);
            components.push_back(component);
        }

    return components;
}

// DAG of strongly connected components
vector<vector<int>> scc_graph(const vector<vector<int>> &graph,
     const vector<vector<int>> &components) {
    vector<int> comp(graph.size());
    for (int i = 0; i < components.size(); i++)
```

```
        for (int u : components[i])
            comp[u] = i;
    vector<vector<int>> g(components.size());
    unordered_set<long long> edges;
    for (int u = 0; u < graph.size(); u++)
        for (int v : graph[u])
            if (comp[u] != comp[v] && edges.insert(((long long)
                 comp[u] << 32) + comp[v]).second)
                g[comp[u]].push_back(comp[v]);
    return g;
}

// usage example
int main() {
    vector<vector<int>> graph = {{1}, {0}, {0, 1}};

    vector<vector<int>> components = scc(graph);
    for (auto &component : components) {
        for (int v : component)
            cout << v << " ";
        cout << endl;
    }

    vector<vector<int>> sg = scc_graph(graph, components);
    for (auto &a : sg) {
        for (int v : a)
            cout << v << " ";
        cout << endl;
    }
}
```

TopologicalSort.cpp

Description: desc

cf6c53, 31 lines

```
// https://en.wikipedia.org/wiki/Topological_sorting

void dfs(const vector<vector<int>> &graph, vector<bool> &used,
     vector<int> &order, int u) {
    used[u] = true;
    for (int v : graph[u])
        if (!used[v])
            dfs(graph, used, order, v);
    order.push_back(u);
}

vector<int> topological_sort(const vector<vector<int>> &graph)
     {
    size_t n = graph.size();
    vector<bool> used(n);
    vector<int> order;
    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs(graph, used, order, i);
    reverse(order.begin(), order.end());
    return order;
}

// usage example
int main() {
    vector<vector<int>> g = {{0}, {}, {0, 1}};

    vector<int> order = topological_sort(g);

    for (int v : order)
        cout << v << " ";
    cout << endl;
}
```

TreeCenters.cpp

Description: desc

aea483, 21 lines

```
// returns vertex that has all its subtrees sizes <= n/2
int find_tree_centroid(const vector<vector<int>> &tree, int u,
     int p) {
    int n = tree.size();
    int cnt = 1;
    bool goodCenter = true;
    for (int v : tree[u]) {
        if (v == p)
            continue;
        int res = find_tree_centroid(tree, v, u);
        if (res >= 0)
            return res;
        int size = -res;
        goodCenter &= size <= n / 2;
        cnt += size;
    }
    goodCenter &= n - cnt <= n / 2;
    return goodCenter ? u : -cnt;
}

// usage example
int main() {}
```

TreeIsomorphism.cpp

Description: desc

13d630, 150 lines

```
vector<vector<int>> children, subtreeLabels, tree, L;
vector<int> pred, mapping;
int n;

auto compare = [](int a, int b) -> bool { return subtreeLabels[
    a] < subtreeLabels[b]; };
auto equals = [](int a, int b) -> bool { return subtreeLabels[a
    ] == subtreeLabels[b]; };

void generate_mapping(int r1, int r2) {
    mapping.resize(n);
    mapping[r1] = r2 - n;
    sort(children[r1].begin(), children[r1].end(), compare);
    sort(children[r2].begin(), children[r2].end(), compare);
    for (int i = 0; i < children[r1].size(); i++) {
        int u = children[r1][i];
        int v = children[r2][i];
        generate_mapping(u, v);
    }
}

vector<int> find_center(int offset = 0) {
    int cnt = n;
    vector<int> a;
    vector<int> deg(n);
    for (int i = 0; i < n; i++) {
        deg[i] = tree[i + offset].size();
        if (deg[i] <= 1) {
            a.push_back(i + offset);
            --cnt;
        }
    }
    while (cnt > 0) {
        vector<int> na;
        for (int i = 0; i < a.size(); i++) {
            int u = a[i];
            for (int j = 0; j < tree[u].size(); j++) {
                int v = tree[u][j];
                if (--deg[v - offset] == 1) {
                    na.push_back(v);
                    --cnt;
                }
            }
        }
        a = na;
    }
    return a;
}

int dfs(int u, int p = -1, int depth = 0) {
    L[depth].push_back(u);
    int h = 0;
    for (int i = 0; i < tree[u].size(); i++) {
        int v = tree[u][i];
        if (v == p)
            continue;
        pred[v] = u;
        children[u].push_back(v);
        h = max(h, dfs(v, u, depth + 1));
    }
    return h + 1;
}

bool rooted_tree_isomorphism(int r1, int r2) {
    L.assign(n, vector<int>());
    pred.assign(2 * n, -1);
    children.assign(2 * n, vector<int>());

    int h1 = dfs(r1);
    int h2 = dfs(r2);
    if (h1 != h2)
        return false;

    int h = h1 - 1;
    vector<int> label(2 * n);
    subtreeLabels.assign(2 * n, vector<int>());

    for (int i = h - 1; i >= 0; i--) {
        for (int j = 0; j < L[i + 1].size(); j++) {
            int v = L[i + 1][j];
            subtreeLabels[pred[v]].push_back(label[v]);
        }

        sort(L[i].begin(), L[i].end(), compare);

        for (int j = 0, cnt = 0; j < L[i].size(); j++) {
            if (j && !equals(L[i][j], L[i][j - 1]))
                ++cnt;
            label[L[i][j]] = cnt;
        }
    }

    if (!equals(r1, r2))
        return false;

    generate_mapping(r1, r2);
    return true;
}

bool treeIsomorphism() {
    vector<int> c1 = find_center();
    vector<int> c2 = find_center(n);
    if (c1.size() == c2.size()) {
        if (rooted_tree_isomorphism(c1[0], c2[0]))
            return true;
        else if (c1.size() > 1)
            return rooted_tree_isomorphism(c1[1], c2[0]);
    }
    return false;
}
```

```
int main() {
    n = 5;
    vector<vector<int>> t1(n);
    t1[0].emplace_back(1);
    t1[1].emplace_back(0);
    t1[1].emplace_back(2);
    t1[2].emplace_back(1);
    t1[1].emplace_back(3);
    t1[3].emplace_back(1);
    t1[0].emplace_back(4);
    t1[4].emplace_back(0);

    vector<vector<int>> t2(n);
    t2[0].emplace_back(1);
    t2[1].emplace_back(0);
    t2[0].emplace_back(4);
    t2[4].emplace_back(0);
    t2[4].emplace_back(3);
    t2[3].emplace_back(4);
    t2[4].emplace_back(2);
    t2[2].emplace_back(4);

    tree.assign(2 * n, vector<int>());
    for (int u = 0; u < n; u++) {
        for (int i = 0; i < t1[u].size(); i++) {
            int v = t1[u][i];
            tree[u].emplace_back(v);
        }
        for (int i = 0; i < t2[u].size(); i++) {
            int v = t2[u][i];
            tree[u + n].emplace_back(v + n);
        }
    }

    bool res = treeIsomorphism();
    cout << res << endl;

    if (res)
        for (int i = 0; i < n; i++)
            cout << mapping[i] << endl;
}
```

linearalgebra (2)

Gauss.cpp

Description: desc

"../numbertheory/modint.h", "matrix.h" a7e7ce, 109 lines

```
vector<double> gauss(vector<vector<double>> A, vector<double> b
    ) {
    int n = A.size();
    for (int i = 0; i < n; i++) {
        int pivot = i;
        for (int j = i + 1; j < n; j++)
            if (abs(A[pivot][i]) < abs(A[j][i]))
                pivot = j;
        swap(A[i], A[pivot]);
        swap(b[i], b[pivot]);
        for (int j = i + 1; j < n; j++)
            A[i][j] /= A[i][i];
        b[i] /= A[i][i];
        for (int j = 0; j < n; j++)
            if (j != i && A[j][i] != 0) {
                for (int k = i + 1; k < n; k++)
                    A[j][k] -= A[i][k] * A[j][i];
                b[j] -= b[i] * A[j][i];
            }
    }
    return b;
}

template <class T>
vector<T> gauss(vector<vector<T>> A, vector<T> b) {
    int n = A.size();
    for (int i = 0; i < n; i++) {
        int pivot = i;
        for (int j = i; j < n; j++)
            if (A[j][i] != 0) {
                pivot = j;
                break;
            }
        assert(A[pivot][i] != 0);
        swap(A[i], A[pivot]);
        swap(b[i], b[pivot]);
        for (int j = i + 1; j < n; j++)
            A[i][j] /= A[i][i];
        b[i] /= A[i][i];
        for (int j = 0; j < n; j++)
            if (j != i && A[j][i] != 0) {
                for (int k = i + 1; k < n; k++)
                    A[j][k] -= A[i][k] * A[j][i];
                b[j] -= b[i] * A[j][i];
            }
    }
    return b;
}

template <class T>
vector<vector<T>> inverse(vector<vector<T>> A) {
    int n = A.size();
    vector<vector<T>> B(n, vector<T>(n));
    for (int i = 0; i < n; ++i)
        B[i][i] = 1;
    for (int i = 0; i < n; i++) {
        int pivot = i;
        for (int j = i; j < n; j++)
            if (A[j][i] != 0) {
                pivot = j;
                break;
            }
        assert(A[pivot][i] != 0);
        swap(A[i], A[pivot]);
        swap(B[i], B[pivot]);
        T Aii = A[i][i];
        for (int j = 0; j < n; j++) {
            A[i][j] /= Aii;
            B[i][j] /= Aii;
        }
        for (int j = 0; j < n; j++) {
            T Aji = A[j][i];
            if (j != i && Aji != 0)
                for (int k = 0; k < n; k++) {
                    A[j][k] -= A[i][k] * Aji;
                    B[j][k] -= B[i][k] * Aji;
                }
        }
    }
    return B;
}

constexpr int mod = (int)1e9 + 7;
using mint = modint<mod>;

// usage example
int main() {
    {
        vector<vector<double>> A{{4, 2, -1}, {2, 4, 3}, {-1, 3,
            5}};
        vector<double> b = {1, 0, 0};
        vector<double> x = gauss(A, b);
        vector<vector<double>> y = A * transpose(vector<vector<
            double>>{x});
        for (int i = 0; i < b.size(); i++)
            assert(abs(b[i] - y[i][0]) < 1e-9);
    }
    {
        vector<vector<mint>> A{{4, 2, -1}, {2, 4, 3}, {-1, 3, 5
            }};
        vector<mint> b = {1, 2, 3};
        vector<mint> x = gauss(A, b);
        vector<vector<mint>> y = A * transpose(vector<vector<
            mint>>{x});
        assert(transpose(y)[0] == b);

        vector<vector<mint>> B = inverse(A);
        vector<vector<mint>> E = A * B;
        for (int i = 0; i < A.size(); ++i)
            for (int j = 0; j < A.size(); ++j)
                assert(E[i][j] == (i == j ? 1 : 0));
    }
}
```

Matrix.cpp

Description: desc

"matrix.h", "../numbertheory/modint.h" 99cd9f, 19 lines

```
constexpr int mod = (int)1e9 + 7;
using mint = modint<mod>;

// usage example
int main() {
    // Fibonacci numbers
    vector<mint> f{1, 1};
    vector<mint> a{1, 1};
    for (int i = 0; i < 60; ++i) {
        cout << (int)nth_element_of_recurrence(a, f, i) << endl
            ;
    }

    vector<vector<mint>> A(2, vector<mint>(2));
    A[0][0] = 1;
    A[0][1] = 1;
    A[1][0] = 1;
    matrix_print(matrix_pow_sum(A, 4));
    matrix_print(A + (A ^ 2) + (A ^ 3) + (A ^ 4));
}
```

Matrix.h

Description: desc

35ec5c, 113 lines

```
template <class T>
vector<vector<T>> matrix_unit(int n) {
    vector<vector<T>> res(n, vector<T>(n));
    for (int i = 0; i < n; i++)
        res[i][i] = 1;
    return res;
}

template <class T>
vector<vector<T>> &operator+=(vector<vector<T>> &a, const
    vector<vector<T>> &b) {
    for (size_t i = 0; i < a.size(); i++)
        for (size_t j = 0; j < a[0].size(); j++)
            a[i][j] += b[i][j];
    return a;
}
```

```
template <class T>
vector<vector<T>> operator+(vector<vector<T>> a, const vector<
    vector<T>> &b) {
    a += b;
    return a;
}

template <class T>
vector<vector<T>> operator*(const vector<vector<T>> &a, const
    vector<vector<T>> &b) {
    int n = a.size();
    int m = a[0].size();
    int k = b[0].size();
    vector<vector<T>> res(n, vector<T>(k));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < k; j++)
            for (int p = 0; p < m; p++)
                res[i][j] += a[i][p] * b[p][j];
    return res;
}

template <class T>
vector<vector<T>> &operator*=(vector<vector<T>> &a, const
    vector<vector<T>> &b) {
    a = a * b;
    return a;
}

template <class T>
vector<vector<T>> operator^(const vector<vector<T>> &a, long
    long p) {
    vector<vector<T>> res = matrix_unit<T>(a.size());
    int highest_one_bit = -1;
    while (1LL << (highest_one_bit + 1) <= p)
        ++highest_one_bit;
    for (int i = highest_one_bit; i >= 0; i--) {
        res *= res;
        if (p >> i & 1) {
            res *= a;
        }
    }
    return res;
}

template <class T>
vector<vector<T>> transpose(const vector<vector<T>> &a) {
    int n = a.size();
    int m = a[0].size();
    vector<vector<T>> b(m, vector<T>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            b[j][i] = a[i][j];
        }
    }
    return b;
}

// a + a^2 + ... + a^p
template <class T>
vector<vector<T>> matrix_pow_sum(const vector<vector<T>> &a,
    long long p) {
    int n = a.size();
    vector<vector<T>> res = vector<vector<T>>(n, vector<T>(n));
    vector<vector<T>> b = matrix_unit<T>(n);
    int highest_one_bit = -1;
    while (1LL << (highest_one_bit + 1) <= p)
        ++highest_one_bit;
    for (int i = highest_one_bit; i >= 0; i--) {
        res = res * (matrix_unit<T>(n) + b);
        b *= b;
        if (p >> i & 1) {
            b *= a;
            res = res * a + a;
        }
    }
    return res;
}

// returns f[n] = f[n-1]*a[k-1] + ... + f[n-k]*a[0], where f
    [0], ..., f[k-1] are provided
// O(k^3*log(n)) complexity
template <class T>
T nth_element_of_recurrence(const vector<T> &a, const vector<T>
    &f, long long n) {
    int k = f.size();
    if (n < k)
        return f[n];
    vector<vector<T>> A(k, vector<T>(k));
    A[k - 1] = a;
    for (int i = 0; i < k - 1; ++i) {
        A[i][i + 1] = 1;
    }
    vector<vector<T>> F = transpose(vector<vector<T>>{f});
    return ((A ^ n) * F)[0][0];
}

template <class T>
void matrix_print(const vector<vector<T>> &a) {
    for (auto &row : a) {
        for (T x : row)
            cout << (int)x << " ";
        cout << endl;
    }
}
```

Tridiagonal.cpp

Description: desc

bd75dc, 26 lines

```
// https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm
template <class T>
vector<T> tridiagonal_solve(vector<T> diag, const vector<T> &
    super, const vector<T> &sub, vector<T> b) {
    for (int i = 0; i < b.size() - 1; ++i) {
        diag[i + 1] -= super[i] * sub[i] / diag[i];
        b[i + 1] -= b[i] * sub[i] / diag[i];
    }
    for (int i = b.size() - 1; i > 0; --i) {
        b[i] /= diag[i];
        b[i - 1] -= b[i] * super[i - 1];
    }
    b[0] /= diag[0];
    return b;
}

// usage example
int main() {
    // -1  1  0   x[0]   5
    //  3 -1  2 * x[1] = 6
    //  0  4 -1   x[2]   7
    vector<double> x = tridiagonal_solve<double>({-1, -1, -1},
        {1, 2}, {3, 4}, {5, 6, 7});

    for (double v : x)
        cout << fixed << setprecision(5) << v << " ";
    cout << endl;
}
```

misc (3)

AllNearestSmallerValues.cpp

Description: desc

48d499, 22 lines

```
// https://en.wikipedia.org/wiki/All_nearest_smaller_values
vector<int> nsv(const vector<int> &a) {
    int n = a.size();
    vector<int> p(n);
    for (int i = 0; i < n; i++) {
        int j = i - 1;
        while (j != -1 && a[j] >= a[i]) {
            j = p[j];
        }
        p[i] = j;
    }
    return p;
}

// usage example
int main() {
    const vector<int> p = nsv({1, 1, 3, 2});

    for (int x : p)
        cout << x << " ";
    cout << endl;
}
```

BinaryExponentiation.cpp

Description: desc

1aec42, 16 lines

```
// https://en.wikipedia.org/wiki/Exponentiation_by_squaring
// https://cp-algorithms.com/algebra/binary-exp.html
int pow_mod(int x, int n, int mod) {
    int res = 1;
    for (long long p = x; n > 0; n >>= 1, p = (p * p) % mod)
        if ((n & 1) != 0)
            res = (int)(res * p % mod);
    return res;
}

// usage example
int main() {
    const int MOD = 1000'000'007;
    int x = pow_mod(2, 10, MOD);
    cout << x << endl;
}
```

BinarySearch.cpp

Description: desc

32e434, 44 lines

```
int binary_search(bool (*f)(int) /* function<bool(int)> f */,
    int from_inclusive, int to_inclusive) {
    // invariant: f[lo] == false, f[hi] == true
    int lo = from_inclusive - 1;
    int hi = to_inclusive + 1;
    // while there are some elements between lo and hi
    while (hi - lo > 1) {
        // invariant: lo < mid < hi
        int mid = (lo + hi) / 2;
        if (!f(mid)) {
            lo = mid;
        } else {
            hi = mid;
        }
    }
    // here lo + 1 == high
    return hi;
}
```

```
// binary_search(new bool[5]{false, false, false, true, true},
    0, 4) == 3
int binary_search(bool a[], int from_inclusive, int
    to_inclusive) {
    // invariant: f[lo] == false, f[hi] == true
    int lo = from_inclusive - 1;
    int hi = to_inclusive + 1;
    // while there are some elements between lo and hi
    while (hi - lo > 1) {
        // invariant: lo < mid < hi
        int mid = (lo + hi) / 2;
        if (!a[mid]) {
            lo = mid;
        } else {
            hi = mid;
        }
    }
    // here lo + 1 == high
    return hi;
}

// usage example
int main() {
    int first_true = binary_search([](int x) { return x >= 4; }
        , 0, 10);
    cout << (first_true == 4) << endl;

    cout << (binary_search(new bool[3]{false, true, true}, 0,
        2) == 1) << endl;
}
```

Comparator.cpp

Description: desc

bd5409, 42 lines

```
struct point {
    int x, y;
};

struct point2 {
    int x, y;

    bool operator<(const point2 &b) const { return x == b.x ? y
        < b.y : x < b.x; }

    bool operator<(int qx) const { return x < qx; }
};

int main() {
    point a[] = {{2, 3}, {1, 2}};

    auto cmp = [](auto &a, auto &b) { return a.x < b.x || (a.x
        == b.x && a.y < b.y); };
    sort(a, a + 2, cmp);

    set<point, decltype(cmp)> s(a, a + 2, cmp);
    for (auto &it : s) {
        cout << it.x << " " << it.y << endl;
    }

    point2 b[] = {{2, 3}, {1, 2}};
    set<point2, less<>> s2(b, b + 2);
    for (auto &it : s2) {
        cout << it.x << " " << it.y << endl;
    }
    cout << s2.lower_bound(1)->y << endl;

    map<point, int, decltype(cmp)> m({{point{2, 3}, 1}}, cmp);
    for (auto &entry : m) {
        cout << entry.first.x << " " << entry.first.y << " " <<
            entry.second << endl;
    }

    priority_queue<point, vector<point>, decltype(cmp)> q(a, a
        + 2, cmp);
    while (!q.empty()) {
        auto item = q.top();
        q.pop();
        cout << item.x << " " << item.y << endl;
    }
}
```

Hashcode.cpp

Description: desc

5218c1, 15 lines

```
struct point {
    int x, y;

    bool operator==(const point &p) const { return x == p.x &&
        y == p.y; }
};

struct hasher {
    size_t operator()(const point &p) const { return p.x * 37 +
        p.y; }
};

int main() {
    unordered_map<point, int, hasher> m;

    m[{1, 2}] = 1;
}
```

HashcodeUnorderedMap.cpp

Description: desc

f01337, 18 lines

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::
            steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

int main() {
    unordered_map<long long, int, custom_hash> safe_map;
}
```

Java8-anti-quicksort-test.cpp

Description: desc

e5e150, 300 lines

```
constexpr int INSERTION_SORT_THRESHOLD = 47;

int MIN_VALUE;
int MAX_VALUE;
constexpr int NO_VALUE = -1;
constexpr int MAX_SIZE = 300'000;
int a[MAX_SIZE];
int p[MAX_SIZE];
int s[MAX_SIZE];

mt19937 rng(1);

inline bool LESS(int a, int b) {
    if (a != NO_VALUE && b != NO_VALUE) {
        return a < b;
    }
    if (a == NO_VALUE) {
        return b > MAX_VALUE;
    }
    return a < MIN_VALUE;
}

inline bool GREATER(int a, int b) {
    if (a != NO_VALUE && b != NO_VALUE) {
        return a > b;
    }
    if (a == NO_VALUE) {
        return b < MIN_VALUE;
    }
    return a > MAX_VALUE;
}

void hackedSort(int left, int right, bool leftmost) {
    int length = right - left + 1;

    // Use insertion sort on tiny arrays
    if (length < INSERTION_SORT_THRESHOLD) {
        for (int i = right; i >= left; i--) {
            if (a[i] == NO_VALUE)
                a[i] = MIN_VALUE++;
        }
        shuffle(a + left, a + right + 1, rng);  // why not?

        if (leftmost) {
            for (int i = left, j = i; i < right; j = ++i) {
                int ai = a[i + 1];
                int pi = p[i + 1];
                while (ai < a[j]) {
                    a[j + 1] = a[j];
                    p[j + 1] = p[j];
                    if (j-- == left) {
                        break;
                    }
                }
                a[j + 1] = ai;
                p[j + 1] = pi;
            }
        } else {
            do {
                if (left >= right) {
                    return;
                }
                ++left;
            } while (a[left] >= a[left - 1]);
            for (int k = left; ++left <= right; k = ++left) {
                int a1 = a[k], a2 = a[left];
                int p1 = p[k], p2 = p[left];

                if (a1 < a2) {
                    a2 = a1;
                    a1 = a[left];
                    p2 = p1;
                    p1 = p[left];
                }
                while (a1 < a[--k]) {
                    a[k + 2] = a[k];
                    p[k + 2] = p[k];
                }
```

```
            ++k;
            a[k + 1] = a1;
            p[k + 1] = p1;

            while (a2 < a[--k]) {
                a[k + 1] = a[k];
                p[k + 1] = p[k];
            }
            a[k + 1] = a2;
            p[k + 1] = p2;
        }
        int last = a[right];
        int plast = p[right];

        while (last < a[--right]) {
            a[right + 1] = a[right];
            p[right + 1] = p[right];
        }
        a[right + 1] = last;
        p[right + 1] = plast;
    }
    return;
}
int seventh = (length >> 3) + (length >> 6) + 1;
int e3 = (left + right) >> 1;
int e2 = e3 - seventh;
int e1 = e2 - seventh;
int e4 = e3 + seventh;
int e5 = e4 + seventh;

if (a[e5] == NO_VALUE)
    a[e5] = MIN_VALUE++;
if (a[e4] == NO_VALUE)
    a[e4] = MIN_VALUE++;

if (a[e1] == NO_VALUE)
    a[e1] = MAX_VALUE--;
if (a[e2] == NO_VALUE)
    a[e2] = MAX_VALUE--;
if (LESS(a[e2], a[e1])) {
    int t = a[e2];
    a[e2] = a[e1];
    a[e1] = t;
    int s = p[e2];
    p[e2] = p[e1];
    p[e1] = s;
}

if (LESS(a[e3], a[e2])) {
    int t = a[e3];
    a[e3] = a[e2];
    a[e2] = t;
    int s = p[e3];
    p[e3] = p[e2];
    p[e2] = s;
    if (LESS(t, a[e1])) {
        a[e2] = a[e1];
        a[e1] = t;
        p[e2] = p[e1];
        p[e1] = s;
    }
}
if (LESS(a[e4], a[e3])) {
    int t = a[e4];
    a[e4] = a[e3];
    a[e3] = t;
    int s = p[e4];
    p[e4] = p[e3];
    p[e3] = s;
    if (LESS(t, a[e2])) {
        a[e3] = a[e2];
        a[e2] = t;
        p[e3] = p[e2];
        p[e2] = s;
        if (LESS(t, a[e1])) {
            a[e2] = a[e1];
            a[e1] = t;
            p[e2] = p[e1];
            p[e1] = s;
        }
    }
}
if (LESS(a[e5], a[e4])) {
    int t = a[e5];
    a[e5] = a[e4];
    a[e4] = t;
    int s = p[e5];
    p[e5] = p[e4];
    p[e4] = s;
    if (LESS(t, a[e3])) {
        a[e4] = a[e3];
        a[e3] = t;
        p[e4] = p[e3];
        p[e3] = s;
        if (LESS(t, a[e2])) {
            a[e3] = a[e2];
            a[e2] = t;
            p[e3] = p[e2];
            p[e2] = s;
            if (LESS(t, a[e1])) {
                a[e2] = a[e1];
                a[e1] = t;
                p[e2] = p[e1];
                p[e1] = s;
            }
        }
    }
}

int less = left;
int great = right;
if (a[e1] != a[e2] && a[e2] != a[e3] && a[e3] != a[e4] && a
     [e4] != a[e5]) {
    int pivot1 = a[e2];
    int pivot2 = a[e4];
    int ppivot1 = p[e2];
    int ppivot2 = p[e4];
    a[e2] = a[left];
    a[e4] = a[right];
    p[e2] = p[left];
    p[e4] = p[right];
    while (LESS(a[++less], pivot1))
        ;
    while (GREATER(a[--great], pivot2))
        ;
    for (int k = less - 1; ++k <= great;) {
        int ak = a[k];
        int pk = p[k];
        if (LESS(ak, pivot1)) {
            a[k] = a[less];
            p[k] = p[less];
            a[less] = ak;
            p[less] = pk;
            ++less;
        } else if (GREATER(ak, pivot2)) {
            while (GREATER(a[great], pivot2)) {
                if (great-- == k) {
                    goto m1;
                }
            }
            if (LESS(a[great], pivot1)) {
                a[k] = a[less];
                p[k] = p[less];
                a[less] = a[great];
                p[less] = p[great];
                ++less;
            } else {
                a[k] = a[great];
                p[k] = p[great];
            }
            a[great] = ak;
            p[great] = pk;
            --great;
        }
    }
    m1:
        a[left] = a[less - 1];
        a[less - 1] = pivot1;
        a[right] = a[great + 1];
        a[great + 1] = pivot2;
        p[left] = p[less - 1];
        p[less - 1] = ppivot1;
        p[right] = p[great + 1];
        p[great + 1] = ppivot2;
        hackedSort(left, less - 2, leftmost);
        hackedSort(great + 2, right, false);
        if (less < e1 && e5 < great) {
            for (int k = less - 1; ++k <= great;) {
                int ak = a[k];
                int pk = p[k];
                if (ak == pivot1) {  // Move a[k] to left part
                    a[k] = a[less];
                    p[k] = p[less];
                    a[less] = ak;
                    p[less] = pk;
                    ++less;
                } else if (ak == pivot2) {  // Move a[k] to
                     right part
                    while (a[great] == pivot2) {
                        if (great-- == k) {
                            goto m2;
                        }
                    }
                    if (a[great] == pivot1) {
                        a[k] = a[less];
                        p[k] = p[less];
                        a[less] = pivot1;
                        p[less] = ppivot1;
                        ++less;
                    } else {
                        a[k] = a[great];
                        p[k] = p[great];
                    }
                    a[great] = ak;
                    p[great] = pk;
                    --great;
                }
            }
        m2:;
        }
        hackedSort(less, great, false);
    }
}

int main() {
    int n = 100’000;
```

```
    for (int i = 0; i < n; i++) {
        a[i] = NO_VALUE;
        p[i] = i;
    }
    MIN_VALUE = 1;
    MAX_VALUE = n;

    hackedSort(0, n - 1, true);

    for (int i = 0; i < n; i++) {
        s[p[i]] = a[i];
    }

    cout << n << endl;
    copy(s, s + n, ostream_iterator<int>(cout, " "));
    cout << endl;
}
```

Knapsack.cpp

Description: desc

1565fd, 34 lines

```
vector<bool> knapsack(const vector<int> &w, int W) {
    vector<int> cnt(W + 1);
    for (int x : w) {
        if (x <= W)
            ++cnt[x];
    }
    vector<bool> can(W + 1);
    can[0] = true;
    for (int v = 1; v <= W; ++v) {
        int am = cnt[v];
        if (am == 0)
            continue;
        for (int from = 0; from < v; ++from) {
            int counter = -1;
            for (int got = from; got <= W; got += v) {
                --counter;
                if (can[got])
                    counter = am;
                else if (counter >= 0) {
                    can[got] = true;
                }
            }
        }
    }
    return can;
}

// usage example
int main() {
    auto b = knapsack({2, 3, 6, 7}, 10);
    for (auto x : b)
        cout << x;
    cout << endl;
}
```

MaxZeroSubMatrix.cpp

Description: desc

70401a, 39 lines

```
int maximum_zero_submatrix(const vector<vector<int>> &a) {
    int R = a.size();
    int C = a[0].size();

    int res = 0;
    vector<int> d(C, -1);
    vector<int> d1(C);
    vector<int> d2(C);
    vector<int> st(C);
    for (int r = 0; r < R; ++r) {
        for (int c = 0; c < C; ++c)
            if (a[r][c] == 1)
                d[c] = r;
        int size = 0;
        for (int c = 0; c < C; ++c) {
            while (size > 0 && d[st[size - 1]] <= d[c])
                --size;
            d1[c] = size == 0 ? -1 : st[size - 1];
            st[size++] = c;
        }
        size = 0;
        for (int c = C - 1; c >= 0; --c) {
            while (size > 0 && d[st[size - 1]] <= d[c])
                --size;
            d2[c] = size == 0 ? C : st[size - 1];
            st[size++] = c;
        }
        for (int j = 0; j < C; ++j)
            res = max(res, (r - d[j]) * (d2[j] - d1[j] - 1));
    }
    return res;
}

// usage example
int main() {
    vector<vector<int>> m{{1, 0, 0, 1}, {1, 0, 0, 1}, {1, 1, 0,
        1}};
    int area = maximum_zero_submatrix(m);
    cout << area << endl;
}
```

Scanner.cpp

Description: desc

<memory.h>, <cstdio> b1b6d8, 50 lines

```
const int BUF_SIZE = 65536;
char input[BUF_SIZE];

struct scanner {
    char* curPos;

    scanner() {
        fread(input, 1, sizeof(input), stdin);
        curPos = input;
    }

    void ensureCapacity() {
        int size = input + BUF_SIZE - curPos;
        if (size < 100) {
            memcpy(input, curPos, size);
            fread(input + size, 1, BUF_SIZE - size, stdin);
            curPos = input;
        }
    }

    int nextInt() {
        ensureCapacity();
        while (*curPos <= ' ')
            ++curPos;
        bool sign = false;
        if (*curPos == '-') {
            sign = true;
            ++curPos;
        }
        int res = 0;
        while (*curPos > ' ')
            res = res * 10 + (*(curPos++) & 15);
        return sign ? -res : res;
    }

    char nextChar() {
        ensureCapacity();
        while (*curPos <= ' ')
            ++curPos;
        return *(curPos++);
    }
};

int main() {
    scanner sc;
    int a = sc.nextInt();
    char b = sc.nextChar();

    printf("%d %c\n", a, b);
}
```

numbertheory (4)

DiscreteLog.cpp

Description: desc

0badf8, 34 lines

```
// returns any x such that a^x = b (mod m)
// O(m^0.5) complexity
int discrete_log(int a, int b, int m) {
    assert(gcd(a, m) == 1);

    int n = (int)sqrt(m) + 1;

    int an = 1;
    for (int i = 0; i < n; ++i)
        an = ((long long)an * a) % m;

    unordered_map<int, int> vals;
    for (int i = 1, cur = an; i <= n; ++i) {
        if (!vals.count(cur))
            vals[cur] = i;
        cur = ((long long)cur * an) % m;
    }

    for (int i = 0, cur = b; i <= n; ++i) {
        if (vals.count(cur)) {
            int res = (long long)vals[cur] * n - i;
            if (res < m)
                return res;
        }
        cur = ((long long)cur * a) % m;
    }
    return -1;
}

// usage example
int main() {
    // 2^x = 3 (mod 5), x = 3
    cout << discrete_log(2, 3, 5) << endl;
}
```

DiscreteRoot.cpp

Description: desc

a6e798, 63 lines

```
// https://cp-algorithms.com/algebra/discrete-root.html

int pow_mod(int x, int n, int mod) {
    int res = 1;
    for (long long p = x; n > 0; n >>= 1, p = (p * p) % mod)
        if ((n & 1) != 0)
            res = (int)(res * p % mod);
    return res;
}
```

```cpp
int calc_generator(int m) {
    if (m == 2)
        return 1;
    vector<int> factors;
    int phi = m - 1;
    int n = phi;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) {
            factors.emplace_back(i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        factors.emplace_back(n);
    for (int res = 2; res <= m; ++res) {
        if (gcd(res, m) != 1)
            continue;
        bool ok = true;
        for (size_t i = 0; i < factors.size() && ok; ++i)
            ok &= pow_mod(res, phi / factors[i], m) != 1;
        if (ok)
            return res;
    }
    return -1;
}

// returns any x such that x^a = b (mod m)
// precondition: m is prime
int discrete_root(int a, int b, int m) {
    if (a == 0)
        return -1;
    int g = calc_generator(m);
    int sq = (int)sqrt(m) + 1;
    vector<pair<int, int>> dec(sq);
    for (int i = 1; i <= sq; ++i)
        dec[i - 1] = {pow_mod(g, (long long)i * sq * b % (m -
            1), m), i};
    sort(dec.begin(), dec.end());
    for (int i = 0; i < sq; ++i) {
        int my = pow_mod(g, (long long)i * b % (m - 1), m) * (
            long long)a % m;
        auto it = lower_bound(dec.begin(), dec.end(), make_pair
            (my, 0));
        if (it != dec.end() && it->first == my) {
            int x = it->second * sq - i;
            int delta = (m - 1) / gcd(b, m - 1);
            return pow_mod(g, x % delta, m);
        }
    }
    return -1;
}

// usage example
int main() {
    cout << discrete_root(3, 3, 5) << endl;
}
```

Euclid.cpp

Description: desc

<optional> c2ffc8, 75 lines

```cpp
// precondition: mod > 1 && gcd(a, mod) = 1
int mod_inverse(int a, int mod) {
    int u = 0, v = 1, m = mod;
    while (a != 0) {
        int t = m / a;
        m -= t * a;
        swap(a, m);
        u -= t * v;
        swap(u, v);
    }
    assert(m == 1);
    return u < 0 ? u + mod : u;
}

// returns { gcd(a,b), x, y } such that gcd(a,b) = a*x + b*y
template <class T>
tuple<T, T, T> euclid(T a, T b) {
    T x = 1, y = 0, x1 = 0, y1 = 1;
    // invariant: a=a_orig*x+b_orig*y, b=a_orig*x1+b_orig*y1
    while (b != 0) {
        T q = a / b;
        T _x1 = x1;
        T _y1 = y1;
        T _b = b;
        x1 = x - q * x1;
        y1 = y - q * y1;
        b = a - q * b;
        x = _x1;
        y = _y1;
        a = _b;
    }
    return a > 0 ? tuple{a, x, y} : tuple{-a, -x, -y};
}

// https://cp-algorithms.com/algebra/linear_congruence_equation
    .html
// solves a * x = b (mod m)
int solve1(int a, int b, int m) {
    int g = gcd(a, m);
    if (b % g)
        return -1;
    a /= g;
    b /= g;
    m /= g;
    int x = (long long)b * mod_inverse(a, m) % m;
    return x;
    // all solutions: x[i]=x+i*m, i=0..g-1
}

// https://cp-algorithms.com/algebra/linear-diophantine-
    equation.html
// solves a * x + b * y = c (mod m)
optional<tuple<int, int>> solve2(int a, int b, int c) {
    auto [g, x0, y0] = euclid(abs(a), abs(b));
    if (c % g != 0)
        return {};
    x0 *= c / g;
    y0 *= c / g;
    return make_optional(tuple{a > 0 ? x0 : -x0, b > 0 ? y0 : -
        y0});
    // all solutions: x=x0+k*b/g, y=y0-k*a/g, k in Z
}

// usage example
int main() {
    auto [gcd, x, y] = euclid(6, 9);
    cout << gcd << " = 6 * " << x << " + 9 * " << y << endl;

    cout << "x=" << solve1(2, 3, 5) << endl;

    auto res = solve2(4, 6, 2);
    if (res) {
        auto [xx, yy] = *res;
        cout << "x=" << xx << " y=" << yy << endl;
    }

    cout << mod_inverse(3, 10) << endl;
}
```

Factorization.cpp

Description: desc

4fd5d8, 23 lines

```cpp
// returns prime_divisor -> power
// O(sqrt(n)) complexity
map<long long, int> factorize(long long n) {
    map<long long, int> factors;
    for (int d = 2; (long long)d * d <= n; d++) {
        while (n % d == 0) {
            ++factors[d];
            n /= d;
        }
    }
    if (n > 1) {
        ++factors[n];
    }
    return factors;
}

// usage example
int main() {
    map<long long int, int> factors = factorize(4 * 3 *
        1000000000039);
    for (auto e : factors) {
        cout << e.first << "^" << e.second << " " << endl;
    }
}
```

LinearRecurrence.cpp

Description: desc

"polynom.h" 2b8a71, 68 lines

```cpp
// returns f[n] = f[n-1]*a[k-1] + ... + f[n-k]*a[0], where f
    [0], ..., f[k-1] are provided
// O(k*log(k)*log(n)) complexity
template <class T>
T nth_element_of_recurrence(vector<T> a, const vector<T> &f,
    long long n) {
    if (n < f.size())
        return f[n];
    a = -a;
    a.emplace_back(1);
    vector<T> xn = power({0, 1}, n, a);
    return inner_product(f.begin(), f.begin() + min(f.size(),
        xn.size()), xn.begin(), T{0});
}

// https://en.wikipedia.org/wiki/Berlekamp%E2%80%93
    Massey_algorithm
template <typename M>
vector<M> berlekamp_massey(const vector<M> &a) {
    int n = a.size();
    vector<M> C(n), B(n);
    C[0] = B[0] = 1;
    M b = 1;
    int L = 0;
    for (int i = 0, m = 1; i < n; ++i) {
        M d = a[i];
        for (int j = 1; j <= L; ++j)
            d = d + C[j] * a[i - j];
        if (d == 0) {
            ++m;
            continue;
        }
        vector<M> T = C;
        M coef = d / b;
        for (int j = m; j < n; ++j)
            C[j] -= coef * B[j - m];
```

```
        if (2 * L > i) {
            ++m;
            continue;
        }
        L = i + 1 - L;
        B = T;
        b = d;
        m = 1;
    }
    C.resize(L + 1);
    C.erase(C.begin());
    reverse(C.begin(), C.end());
    return -C;
}

// usage example
constexpr int mod = (int)1e9 + 7;
using mint = modint<mod>;

int main() {
    {
        // Fibonacci numbers
        vector<mint> f{1, 1};
        vector<mint> a{1, 1};
        for (int i = 0; i < 10; ++i) {
            cout << (int)nth_element_of_recurrence(a, f, i) <<
                endl;
        }
        cout << endl;
    }
    {
        vector<mint> f = berlekamp_massey(vector<mint>({1, 1,
            3, 5, 11}));
        for (auto v : f)
            cout << (int)v << " ";
        cout << endl;
    }
}
```

Modint.cpp

Description: desc

"modint.h" afddbe, 12 lines

```
constexpr int mod = (int)1e9 + 7;
using mint = modint<mod>;

// usage example
int main() {
    mint a = 1;
    mint b = 1;
    b += 1;
    mint c = 1000'000'000;
    mint d = a / b * c / c;
    cout << ((int)d) << endl;
}
```

Modint.h

Description: desc

7aa503, 62 lines

```
template <int mod>
struct modint {
    int value;

    modint(long long x = 0) { value = normalize(x); }

    int normalize(long long x) {
        if (x < -mod || x >= mod)
            x %= mod;
        if (x < 0)
            x += mod;
        return static_cast<int>(x);
    }

    explicit operator int() const { return value; }

    modint operator-() const { return modint(-value); }

    modint &operator+=(modint rhs) {
        if ((value += rhs.value) >= mod)
            value -= mod;
        return *this;
    }

    modint &operator-=(modint rhs) {
        if ((value -= rhs.value) < 0)
            value += mod;
        return *this;
    }

    modint &operator*=(modint rhs) {
        value = normalize(static_cast<long long>(value) * rhs.
            value);
        return *this;
    }

    modint &operator/=(modint rhs) { return *this *= modint(
        inverse(rhs.value, mod)); }

    int inverse(int a, int m) {
        int u = 0, v = 1;
        while (a != 0) {
            int t = m / a;
            m -= t * a;
            swap(a, m);
            u -= t * v;
            swap(u, v);
        }
        assert(m == 1);
        return u;
    }

    bool operator==(modint rhs) const { return value == rhs.
        value; }

    bool operator!=(modint rhs) const { return !(*this == rhs);
         }

    friend modint operator+(modint lhs, modint rhs) { return
        lhs += rhs; }

    friend modint operator-(modint lhs, modint rhs) { return
        lhs -= rhs; }

    friend modint operator*(modint lhs, modint rhs) { return
        lhs *= rhs; }

    friend modint operator/(modint lhs, modint rhs) { return
        lhs /= rhs; }
};
```

Polynom.cpp

Description: desc

"polynom.h" e4cd25, 49 lines

```
// usage example
constexpr int mod = (int)1e9 + 7;
using mint = modint<mod>;

int main() {
    {
        vector<mint> poly{3, 2, 1};
        vector<mint> x{1, 2, 3, 4, 5};
        vector<mint> values = evaluate(poly, x);
        for (size_t i = 0; i < x.size(); ++i) {
            cout << (int)values[i] << " " << (int)evaluate(poly
                , x[i]) << endl;
        }
        cout << endl;
    }
    {
        vector<mint> x{7, 2, 1, 3, 5, 6};
        vector<mint> y;
        for (size_t i = 0; i < x.size(); ++i) {
            y.emplace_back(x[i] * x[i] * x[i] + 5 * x[i] * x[i]
                + x[i] + 3);
        }
        vector<mint> poly = interpolate(x, y);
        for (size_t i = 0; i < poly.size(); ++i) {
            cout << (int)poly[i] << " ";
        }
        cout << endl << endl;
    }
    {
        for (int i = 0; i < 10; ++i) {
            vector<mint> s = sequence<mint>(i);
            for (auto x : s)
                cout << (int)x << " ";
            cout << endl;
        }
        cout << endl;
    }
    {
        for (int i = 0; i < 10; ++i) {
            cout << (int)factorial<mint>(i) << endl;
        }

        auto t1 = chrono::high_resolution_clock::now();
        cout << (int)factorial<mint>(1000'000'000) << endl;
        auto t2 = chrono::high_resolution_clock::now();
        chrono::duration<double, milli> duration = t2 - t1;
        cout << duration.count() << " ms" << endl;

        cout << endl;
    }
}
```

Polynom.h

Description: desc

"../numeric/fft.h", "modint.h" 1d064a, 484 lines

```
// https://cp-algorithms.com/algebra/polynomial.html

template <class T>
vector<T> &operator+=(vector<T> &a, const vector<T> &b) {
    if (a.size() < b.size()) {
        a.resize(b.size());
    }
    for (size_t i = 0; i < b.size(); i++) {
        a[i] += b[i];
    }
    return a;
}

template <class T>
vector<T> operator+(vector<T> a, const vector<T> &b) {
    a += b;
    return a;
}

template <class T>
```

```
vector<T> &operator-=(vector<T> &a, const vector<T> &b) {
    if (a.size() < b.size()) {
        a.resize(b.size());
    }
    for (int i = 0; i < b.size(); i++) {
        a[i] -= b[i];
    }
    return a;
}

template <class T>
vector<T> operator-(vector<T> a, const vector<T> &b) {
    a -= b;
    return a;
}

template <class T>
vector<T> operator-(vector<T> a) {
    for (int i = 0; i < a.size(); i++) {
        a[i] = -a[i];
    }
    return a;
}

// fast multiplication for modint in O(n*log(n))
template <int mod>
vector<modint<mod>> operator*(const vector<modint<mod>> &a,
     const vector<modint<mod>> &b) {
    if (a.empty() || b.empty()) {
        return {};
    }
    if (min(a.size(), b.size()) < 150) {
        vector<modint<mod>> c(a.size() + b.size() - 1, 0);
        for (size_t i = 0; i < a.size(); i++) {
            for (size_t j = 0; j < b.size(); j++) {
                c[i + j] += a[i] * b[j];
            }
        }
        return c;
    }
    vector<int> a_int(a.size());
    for (size_t i = 0; i < a.size(); i++) {
        a_int[i] = static_cast<int>(a[i]);
    }
    vector<int> b_int(b.size());
    for (size_t i = 0; i < b.size(); i++) {
        b_int[i] = static_cast<int>(b[i]);
    }
    vector<int> c_int = multiply_mod(a_int, b_int, mod);
    vector<modint<mod>> c(c_int.size());
    for (size_t i = 0; i < c.size(); i++) {
        c[i] = c_int[i];
    }
    return c;
}

// fallback multiplication in O(n*m)
template <class T>
vector<T> operator*(const vector<T> &a, const vector<T> &b) {
    if (a.empty() || b.empty()) {
        return {};
    }
    vector<T> c(a.size() + b.size() - 1, 0);
    for (int i = 0; i < a.size(); i++) {
        for (int j = 0; j < b.size(); j++) {
            c[i + j] += a[i] * b[j];
        }
    }
    return c;
}

template <class T>
vector<T> &operator*=(vector<T> &a, const vector<T> &b) {
    a = a * b;
    return a;
}

template <class T>
vector<T> inverse(const vector<T> &a) {
    assert(!a.empty());
    int n = a.size();
    vector<T> b = {1 / a[0]};
    while (b.size() < n) {
        vector<T> a_cut(a.begin(), a.begin() + min(a.size(), b.
             size() << 1));
        vector<T> x = b * b * a_cut;
        b.resize(b.size() << 1);
        for (size_t i = b.size() >> 1; i < min(x.size(), b.size
             ()); i++) {
            b[i] = -x[i];
        }
    }
    b.resize(n);
    return b;
}

template <class T>
vector<T> &operator/=(vector<T> &a, vector<T> b) {
    int n = a.size();
    int m = b.size();
    if (n < m) {
        a.clear();
    } else {
        reverse(a.begin(), a.end());
        reverse(b.begin(), b.end());
        b.resize(n - m + 1);
        a *= inverse(b);
        a.erase(a.begin() + n - m + 1, a.end());
        reverse(a.begin(), a.end());
    }
    return a;
}

template <class T>
vector<T> operator/(vector<T> a, const vector<T> &b) {
    a /= b;
    return a;
}

template <class T>
vector<T> &operator%=(vector<T> &a, const vector<T> &b) {
    int n = a.size();
    int m = b.size();
    if (n >= m) {
        vector<T> c = (a / b) * b;
        a.resize(m - 1);
        for (int i = 0; i < m - 1; i++) {
            a[i] -= c[i];
        }
    }
    return a;
}

template <class T>
vector<T> operator%(vector<T> a, const vector<T> &b) {
    a %= b;
    return a;
}

template <class T>
vector<T> power(const vector<T> &a, long long b, const vector<T
     > &mod) {
    assert(b >= 0);
    vector<T> res = vector<T>{1} % mod;
    int highest_one_bit = -1;
    while (1LL << (highest_one_bit + 1) <= b)
        ++highest_one_bit;
    for (int i = highest_one_bit; i >= 0; i--) {
        res = res * res % mod;
        if (b >> i & 1) {
            res = res * a % mod;
        }
    }
    return res;
}

template <class T>
vector<T> derivative(vector<T> a) {
    for (size_t i = 0; i < a.size(); i++) {
        a[i] *= i;
    }
    if (!a.empty()) {
        a.erase(a.begin());
    }
    return a;
}

template <class T>
vector<T> integrate(vector<T> a) {
    a.insert(a.begin(), 0);
    for (int i = 1; i < a.size(); i++) {
        a[i] /= i;
    }
    return a;
}

template <class T>
vector<T> logarithm(const vector<T> &a) {
    assert(!a.empty() && a[0] == 1);
    vector<T> res = integrate(derivative(a) * inverse(a));
    res.resize(a.size());
    return res;
}

template <class T>
vector<T> exponent(const vector<T> &a) {
    assert(!a.empty() && a[0] == 0);
    int n = a.size();
    vector<T> b = {1};
    while (b.size() < n) {
        vector<T> x(a.begin(), a.begin() + min(a.size(), b.size
             () << 1));
        x[0] += 1;
        vector<T> old_b = b;
        b.resize(b.size() << 1);
        x -= logarithm(b);
        x *= old_b;
        for (int i = b.size() >> 1; i < min(x.size(), b.size())
             ; i++) {
            b[i] = x[i];
        }
    }
    b.resize(n);
    return b;
}

template <class T>
```

```
vector<T> sqrt(const vector<T> &a) {
    assert(!a.empty() && a[0] == 1);
    int n = a.size();
    vector<T> b = {1};
    while (b.size() < n) {
        vector<T> x(a.begin(), a.begin() + min(a.size(), b.size
            () << 1));
        b.resize(b.size() << 1);
        x *= inverse(b);
        T inv2 = 1 / static_cast<T>(2);
        for (int i = b.size() >> 1; i < min(x.size(), b.size())
            ; i++) {
            b[i] = x[i] * inv2;
        }
    }
    b.resize(n);
    return b;
}

template <class T>
vector<T> multiply(const vector<vector<T>> &a) {
    if (a.empty()) {
        return {0};
    }
    function<vector<T>(int, int)> mult = [&](int l, int r) {
        if (l == r) {
            return a[l];
        }
        int m = (l + r) >> 1;
        return mult(l, m) * mult(m + 1, r);
    };
    return mult(0, (int)a.size() - 1);
}

template <class T>
T evaluate(const vector<T> &a, const T &x) {
    T res = 0;
    for (int i = (int)a.size() - 1; i >= 0; i--) {
        res = res * x + a[i];
    }
    return res;
}

template <class T>
vector<T> evaluate(const vector<T> &a, const vector<T> &x) {
    if (x.empty()) {
        return {};
    }
    if (a.empty()) {
        return vector<T>(x.size());
    }
    int n = x.size();
    vector<vector<T>> t(2 * n - 1);
    function<void(int, int, int)> build = [&](int v, int l, int
         r) {
        if (l == r) {
            t[v] = vector<T>{-x[l], 1};
        } else {
            int m = (l + r) >> 1;
            int y = v + ((m - l + 1) << 1);
            build(v + 1, l, m);
            build(y, m + 1, r);
            t[v] = t[v + 1] * t[y];
        }
    };
    build(0, 0, n - 1);
    vector<T> res(n);
    function<void(int, int, int, vector<T>)> eval = [&](int v,
         int l, int r, vector<T> p) {
        p %= t[v];
        if (p.size() < 150) {
            for (int i = l; i <= r; i++) {
                res[i] = evaluate(p, x[i]);
            }
            return;
        }
        if (l == r) {
            res[l] = p[0];
        } else {
            int m = (l + r) >> 1;
            int z = v + ((m - l + 1) << 1);
            eval(v + 1, l, m, p);
            eval(z, m + 1, r, p);
        }
    };
    eval(0, 0, n - 1, a);
    return res;
}

template <class T>
vector<T> interpolate(const vector<T> &x, const vector<T> &y) {
    if (x.empty()) {
        return {};
    }
    assert(x.size() == y.size());
    int n = x.size();
    vector<vector<T>> t(2 * n - 1);
    function<void(int, int, int)> build = [&](int v, int l, int
         r) {
        if (l == r) {
            t[v] = vector<T>{-x[l], 1};
        } else {
            int m = (l + r) >> 1;
            int z = v + ((m - l + 1) << 1);
            build(v + 1, l, m);
            build(z, m + 1, r);
            t[v] = t[v + 1] * t[z];
        }
    };
    build(0, 0, n - 1);
    vector<T> val(n);
    function<void(int, int, int, vector<T>)> eval = [&](int v,
         int l, int r, vector<T> p) {
        p %= t[v];
        if (p.size() < 150) {
            for (int i = l; i <= r; i++) {
                val[i] = evaluate(p, x[i]);
            }
            return;
        }
        if (l == r) {
            val[l] = p[0];
        } else {
            int m = (l + r) >> 1;
            int z = v + ((m - l + 1) << 1);
            eval(v + 1, l, m, p);
            eval(z, m + 1, r, p);
        }
    };
    vector<T> d = derivative(t[0]);
    eval(0, 0, n - 1, d);
    for (int i = 0; i < n; i++) {
        val[i] = y[i] / val[i];
    }
    function<vector<T>(int, int, int)> calc = [&](int v, int l,
         int r) {
        if (l == r) {
            return vector<T>{val[l]};
        }
        int m = (l + r) >> 1;
        int z = v + ((m - l + 1) << 1);
        return calc(v + 1, l, m) * t[z] + calc(z, m + 1, r) * t
            [v + 1];
    };
    return calc(0, 0, n - 1);
}

// f[i] = 1^n + 2^n + ... + up^n
// O(n*log(n)) complexity
template <class T>
vector<T> faulhaber(const T &up, int n) {
    vector<T> ex(n + 1);
    T e = 1;
    for (int i = 0; i <= n; i++) {
        ex[i] = e;
        e /= i + 1;
    }
    vector<T> den = ex;
    den.erase(den.begin());
    for (auto &d : den) {
        d = -d;
    }
    vector<T> num(n);
    T p = 1;
    for (int i = 0; i < n; i++) {
        p *= up + 1;
        num[i] = ex[i + 1] * (1 - p);
    }
    vector<T> res = num * inverse(den);
    res.resize(n);
    T f = 1;
    for (int i = 0; i < n; i++) {
        res[i] *= f;
        f *= i + 1;
    }
    return res;
}

// (x + 1) * (x + 2) * ... * (x + n)
// (can be optimized with precomputed inverses)
template <class T>
vector<T> sequence(int n) {
    if (n == 0) {
        return {1};
    }
    if (n % 2 == 1) {
        return sequence<T>(n - 1) * vector<T>{n, 1};
    }
    vector<T> c = sequence<T>(n / 2);
    vector<T> a = c;
    reverse(a.begin(), a.end());
    T f = 1;
    for (int i = n / 2 - 1; i >= 0; i--) {
        f *= n / 2 - i;
        a[i] *= f;
    }
    vector<T> b(n / 2 + 1);
    b[0] = 1;
    for (int i = 1; i <= n / 2; i++) {
        b[i] = b[i - 1] * (n / 2) / i;
    }
    vector<T> h = a * b;
    h.resize(n / 2 + 1);
    reverse(h.begin(), h.end());
    f = 1;
    for (int i = 1; i <= n / 2; i++) {
        f /= i;
```

```
        h[i] *= f;
    }
    vector<T> res = c * h;
    return res;
}

template <class T>
T factorial(long long n) {
    if (n == 0)
        return 1;
    int m = min((long long)(sqrt(n) * 2), n);
    vector<T> a = sequence<T>(m);
    vector<T> x(n / m);
    for (size_t i = 0; i < x.size(); ++i) {
        x[i] = i;
        x[i] *= m;
    }
    vector<T> b = evaluate(a, x);
    T res = 1;
    for (auto v : b)
        res *= v;
    for (long long i = n / m * m + 1; i <= n; ++i) {
        res *= i;
    }
    return res;
}

template <class T>
T binomial(int n, int m) {
    return factorial<T>(n) / factorial<T>(m) / factorial<T>(n -
         m);
}

template <class T>
class OnlineProduct {
   public:
    const vector<T> a;
    vector<T> b;
    vector<T> c;

    OnlineProduct(const vector<T> &a_) : a(a_) {}

    T add(const T &val) {
        int i = (int)b.size();
        b.push_back(val);
        if ((int)c.size() <= i) {
            c.resize(i + 1);
        }
        c[i] += a[0] * b[i];
        int z = 1;
        while ((i & (z - 1)) == z - 1 && (int)a.size() > z) {
            vector<T> a_mul(a.begin() + z, a.begin() + min(z <<
                 1, (int)a.size()));
            vector<T> b_mul(b.end() - z, b.end());
            vector<T> c_mul = a_mul * b_mul;
            if ((int)c.size() <= i + (int)c_mul.size()) {
                c.resize(i + c_mul.size() + 1);
            }
            for (int j = 0; j < (int)c_mul.size(); j++) {
                c[i + 1 + j] += c_mul[j];
            }
            z <<= 1;
        }
        return c[i];
    }
};
```

PrimesAndDivisors.cpp

Description: desc

2aae08, 87 lines

```
// https://cp-algorithms.com/algebra/sieve-of-eratosthenes.html

vector<int> get_primes(int n) {
    if (n <= 1)
        return {};
    vector<bool> prime(n + 1, true);
    prime[0] = prime[1] = false;

    for (int i = 2; i * i <= n; i++)
        if (prime[i])
            for (int j = i * i; j <= n; j += i)
                prime[j] = false;

    vector<int> primes;
    for (int i = 0; i < prime.size(); ++i)
        if (prime[i])
            primes.push_back(i);

    return primes;
}

// Generates prime numbers up to n in O(n) time
vector<int> generate_primes_linear_time(int n) {
    vector<int> lp(n + 1);
    vector<int> primes;
    for (int i = 2; i <= n; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            primes.push_back(i);
        }
        for (int j = 0; j < primes.size() && primes[j] <= lp[i]
             && i * primes[j] <= n; ++j)
            lp[i * primes[j]] = primes[j];
    }
    return primes;
}

vector<int> number_of_prime_divisors(int n) {
    vector<int> divisors(n + 1);
    fill(divisors.begin() + 2, divisors.end(), 1);
    for (int i = 2; i * i <= n; ++i)
        if (divisors[i] == 1)
            for (int j = i; j * i <= n; j++)
                divisors[i * j] = divisors[j] + 1;
    return divisors;
}

// Generates minimum prime divisor of all numbers up to n in O(
    n) time
vector<int> generate_min_divisors(int n) {
    vector<int> lp(n + 1);
    lp[1] = 1;
    vector<int> primes;
    for (int i = 2; i <= n; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            primes.push_back(i);
        }
        for (int j = 0; j < primes.size() && primes[j] <= lp[i]
             && i * primes[j] <= n; ++j)
            lp[i * primes[j]] = primes[j];
    }
    return lp;
}

// Generates prime divisor of all numbers up to n

vector<int> generate_divisors(int n) {
    vector<int> divisors(n + 1);
    iota(divisors.begin(), divisors.end(), 0);
    for (int i = 2; i * i <= n; i++)
        if (divisors[i] == i)
            for (int j = i * i; j <= n; j += i)
                divisors[j] = i;
    return divisors;
}

// usage example
int main() {
    auto print = [](const vector<int> &a) {
        for (int x : a)
            cout << x << " ";
        cout << endl;
    };

    int n = 32;
    print(get_primes(n));
    print(generate_primes_linear_time(n));
    print(generate_min_divisors(n));
    print(generate_divisors(n));
}
```

PrimitiveRoot.cpp

Description: desc

4cabe8, 59 lines

```
// https://cp-algorithms.com/algebra/primitive-root.html

int pow_mod(int x, int n, int mod) {
    int res = 1;
    for (long long p = x; n > 0; n >>= 1, p = (p * p) % mod)
        if ((n & 1) != 0)
            res = (int)(res * p % mod);
    return res;
}

int totient_function(int n) {
    int res = n;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            res -= res / i;
        }
    if (n > 1)
        res -= res / n;
    return res;
}

// returns g such that g^i runs through all numbers from 1 to m
    -1 modulo m
// g exists for m = 2,4,p^a,2*p^a, where p > 2 is a prime
    number
// O(m^0.5) complexity
int calc_generator(int m) {
    if (m == 2)
        return 1;
    vector<int> factors;
    int phi = totient_function(m);
    int n = phi;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) {
            factors.emplace_back(i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
```

```
        factors.emplace_back(n);
    for (int res = 2; res <= m; ++res) {
        if (gcd(res, m) != 1)
            continue;
        bool ok = true;
        for (size_t i = 0; i < factors.size() && ok; ++i)
            ok &= pow_mod(res, phi / factors[i], m) != 1;
        if (ok)
            return res;
    }
    return -1;
}

// usage example
int main() {
    for (int i = 0; i < 15; ++i) {
        cout << "generator(" << i << ") = " << calc_generator(i
            ) << endl;
    }
    cout << "generator(" << 998244353 << ") = " <<
        calc_generator(998244353) << endl;
}
```

Rational.cpp

Description: desc

"rational.h" 320125, 12 lines

```
using rt = rational<long long /*__int128*/>;

// usage example
int main() {
    rt x{1, 2};
    rt y{2, 3};
    if (y > x)
        cout << "y > x" << endl;
    x -= y;
    x = x.abs();
    cout << x.a << "/" << x.b << endl;
}
```

Rational.h

Description: desc

eff5c5, 70 lines

```
template <class T>
struct rational {
    T a, b;

    rational(T a, T b = 1) : a{a}, b{b} { normalize(); }

    void normalize() {
        if (b < 0) {
            a = -a;
            b = -b;
        }
        auto g = gcd(::abs(a), b);
        if (g != 0) {
            a /= g;
            b /= g;
        }
    }

    bool operator==(rational rhs) const { return a == rhs.a &&
        b == rhs.b; }

    bool operator!=(rational rhs) const { return !(*this == rhs
        ); }

    bool operator<(rational rhs) const { return a * rhs.b < b *
        rhs.a; }

    bool operator<=(rational rhs) const { return !(rhs < *this)
        ; }

    bool operator>(rational rhs) const { return rhs < *this; }

    bool operator>=(rational rhs) const { return !(*this < rhs)
        ; }

    rational operator-() const { return rational(-a, b); }

    rational abs() const { return rational(::abs(a), b); }

    rational &operator+=(rational rhs) {
        a = a * rhs.b + b * rhs.a;
        b *= rhs.b;
        normalize();
        return *this;
    }

    rational &operator-=(rational rhs) {
        a = a * rhs.b - b * rhs.a;
        b *= rhs.b;
        normalize();
        return *this;
    }

    rational &operator*=(rational rhs) {
        a *= rhs.a;
        b *= rhs.b;
        normalize();
        return *this;
    }

    rational &operator/=(rational rhs) {
        a *= rhs.b;
        b *= rhs.a;
        normalize();
        return *this;
    }

    friend rational operator+(rational lhs, rational rhs) {
        return lhs += rhs; }

    friend rational operator-(rational lhs, rational rhs) {
        return lhs -= rhs; }

    friend rational operator*(rational lhs, rational rhs) {
        return lhs *= rhs; }

    friend rational operator/(rational lhs, rational rhs) {
        return lhs /= rhs; }
};
```

SchreierSims.cpp

Description: desc

019312, 115 lines

```
// https://en.wikipedia.org/wiki/Schreier%E2%80%93
    Sims_algorithm

// time: O(n^2 lg^3 |G| + t n lg |G|)
// mem : O(n^2 lg |G| + tn)
// t : number of generator
namespace SchreierSimsAlgorithm {
using vi = vector<int>;
using pii = pair<int, int>;

vi inv(const vi &p) {
    vi res(p.size());
    for (int i = 0; i < p.size(); i++) {
        res[p[i]] = i;
    }
    return res;
}

vi operator*(const vi &a, const vi &b) {
    vi res(a.size());
    for (int i = 0; i < a.size(); i++)
        res[i] = b[a[i]];
    return res;
}

int n, m;
vector<vector<vector<int>>> bkts, bktsInv;
vector<vector<int>> lookup;

int fast_filter(const vi &g, bool addToG = 1) {
    n = bkts.size();
    vi p = g;
    for (int i = 0; i < n; i++) {
        int res = lookup[i][p[i]];
        if (res == -1) {
            if (addToG) {
                bkts[i].push_back(p);
                bktsInv[i].push_back(inv(p));
                lookup[i][p[i]] = (int)bkts[i].size() - 1;
            }
            return i;
        }
        p = p * bktsInv[i][res];
    }
    return -1;
}

long long calc_total_size() {
    long long ret = 1;
    for (int i = 0; i < n; i++)
        ret *= bkts[i].size();
    return ret;
}

bool in_group(const vi &g) {
    return fast_filter(g, false) == -1;
}

void solve(const vector<vector<int>> &perms, int _n) {
    n = _n;
    m = perms.size();
    bkts.clear();
    bktsInv.clear();
    lookup.clear();
    lookup.resize(n);
    for (int i = 0; i < n; i++) {
        lookup[i].resize(n);
        fill(lookup[i].begin(), lookup[i].end(), -1);
    }
    vi id(n);
    iota(id.begin(), id.end(), 0);
    bkts.resize(n);
    bktsInv.resize(n);
    for (int i = 0; i < n; i++) {
        bkts[i].push_back(id);
        bktsInv[i].push_back(id);
        lookup[i][i] = 0;
    }
    for (int i = 0; i < m; i++)
        fast_filter(perms[i]);
    queue<pair<pii, pii>> toUpd;
    for (int i = 0; i < n; i++)
```

```
        for (int j = i; j < n; j++)
            for (int k = 0; k < bkts[i].size(); k++)
                for (int l = 0; l < bkts[j].size(); l++)
                    toUpd.push({pii(i, k), pii(j, l)});
    while (!toUpd.empty()) {
        pii a = toUpd.front().first;
        pii b = toUpd.front().second;
        toUpd.pop();
        int res = fast_filter(bkts[a.first][a.second] * bkts[b.
            first][b.second]);
        if (res == -1)
            continue;
        pii newPair(res, (int)bkts[res].size() - 1);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < bkts[i].size(); ++j) {
                if (i <= res)
                    toUpd.push({pii(i, j), newPair});
                if (res <= i)
                    toUpd.push({newPair, pii(i, j)});
            }
    }
}
}  // namespace SchreierSimsAlgorithm

// usage example
int main() {
    vector<vector<int>> perms{{0, 1, 3, 2}, {1, 0, 2, 3}};

    SchreierSimsAlgorithm::solve(perms, 4);
    cout << SchreierSimsAlgorithm::in_group({0, 1, 2, 3}) <<
        endl;
    cout << SchreierSimsAlgorithm::in_group({1, 0, 2, 3}) <<
        endl;
    cout << SchreierSimsAlgorithm::in_group({0, 1, 3, 2}) <<
        endl;
    cout << SchreierSimsAlgorithm::in_group({1, 0, 3, 2}) <<
        endl;
    cout << SchreierSimsAlgorithm::in_group({2, 3, 0, 1}) <<
        endl;
}
```

numeric (5)

Bigint.cpp

Description: desc

"fft.h" 52f441, 410 lines

```
constexpr int digits(int base) noexcept {
    return base <= 1 ? 0 : 1 + digits(base / 10);
}

constexpr int base = 1000'000'000;
constexpr int base_digits = digits(base);

constexpr int fft_base = 10'000;  // fft_base^2 * n /
    fft_base_digits <= 10^15 for double
constexpr int fft_base_digits = digits(fft_base);

struct bigint {
    // value == 0 is represented by empty z
    vector<int> z;  // digits

    // sign == 1 <==> value >= 0
    // sign == -1 <==> value < 0
    int sign;

    bigint(long long v = 0) { *this = v; }

    bigint &operator=(long long v) {
        sign = v < 0 ? -1 : 1;
        v *= sign;
        z.clear();
        for (; v > 0; v = v / base)
            z.push_back((int)(v % base));
        return *this;
    }

    bigint(const string &s) { read(s); }

    bigint &operator+=(const bigint &other) {
        if (sign == other.sign) {
            for (int i = 0, carry = 0; i < other.z.size() ||
                carry; ++i) {
                if (i == z.size())
                    z.push_back(0);
                z[i] += carry + (i < other.z.size() ? other.z[i
                    ] : 0);
                carry = z[i] >= base;
                if (carry)
                    z[i] -= base;
            }
        } else if (other != 0 /* prevent infinite loop */) {
            *this -= -other;
        }
        return *this;
    }

    friend bigint operator+(bigint a, const bigint &b) {
        a += b;
        return a;
    }

    bigint &operator-=(const bigint &other) {
        if (sign == other.sign) {
            if ((sign == 1 && *this >= other) || (sign == -1 &&
                *this <= other)) {
                for (int i = 0, carry = 0; i < other.z.size()
                    || carry; ++i) {
                    z[i] -= carry + (i < other.z.size() ? other
                        .z[i] : 0);
                    carry = z[i] < 0;
                    if (carry)
                        z[i] += base;
                }
                trim();
            } else {
                *this = other - *this;
                this->sign = -this->sign;
            }
        } else {
            *this += -other;
        }
        return *this;
    }

    friend bigint operator-(bigint a, const bigint &b) {
        a -= b;
        return a;
    }

    bigint &operator*=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < z.size() || carry; ++i)
            {
            if (i == z.size())
                z.push_back(0);
            long long cur = (long long)z[i] * v + carry;
            carry = (int)(cur / base);
            z[i] = (int)(cur % base);
        }
        trim();
        return *this;
    }

    bigint operator*(int v) const { return bigint(*this) *= v;
        }

    friend pair<bigint, bigint> divmod(const bigint &a1, const
        bigint &b1) {
        int norm = base / (b1.z.back() + 1);
        bigint a = a1.abs() * norm;
        bigint b = b1.abs() * norm;
        bigint q, r;
        q.z.resize(a.z.size());

        for (int i = (int)a.z.size() - 1; i >= 0; i--) {
            r *= base;
            r += a.z[i];
            int s1 = b.z.size() < r.z.size() ? r.z[b.z.size()]
                : 0;
            int s2 = b.z.size() - 1 < r.z.size() ? r.z[b.z.size
                () - 1] : 0;
            int d = (int)(((long long)s1 * base + s2) / b.z.
                back());
            r -= b * d;
            while (r < 0)
                r += b, --d;
            q.z[i] = d;
        }

        q.sign = a1.sign * b1.sign;
        r.sign = a1.sign;
        q.trim();
        r.trim();
        return {q, r / norm};
    }

    friend bigint sqrt(const bigint &a1) {
        bigint a = a1;
        while (a.z.empty() || a.z.size() % 2 == 1)
            a.z.push_back(0);

        int n = a.z.size();

        int firstDigit = (int)::sqrt((double)a.z[n - 1] * base
            + a.z[n - 2]);
        int norm = base / (firstDigit + 1);
        a *= norm;
        a *= norm;
        while (a.z.empty() || a.z.size() % 2 == 1)
            a.z.push_back(0);

        bigint r = (long long)a.z[n - 1] * base + a.z[n - 2];
        firstDigit = (int)::sqrt((double)a.z[n - 1] * base + a.
            z[n - 2]);
        int q = firstDigit;
        bigint res;

        for (int j = n / 2 - 1; j >= 0; j--) {
            for (;; --q) {
                bigint r1 = (r - (res * 2 * base + q) * q) *
                    base * base +
                        (j > 0 ? (long long)a.z[2 * j - 1]
                            * base + a.z[2 * j - 2] : 0);
                if (r1 >= 0) {
                    r = r1;
```

```
                break;
            }
        }
        res *= base;
        res += q;

        if (j > 0) {
            int d1 = res.z.size() + 2 < r.z.size() ? r.z[
                res.z.size() + 2] : 0;
            int d2 = res.z.size() + 1 < r.z.size() ? r.z[
                res.z.size() + 1] : 0;
            int d3 = res.z.size() < r.z.size() ? r.z[res.z.
                size()] : 0;
            q = (int)(((long long)d1 * base * base + (long
                long)d2 * base + d3) / (firstDigit * 2));
        }
    }

    res.trim();
    return res / norm;
}

bigint operator/(const bigint &v) const { return divmod(*
    this, v).first; }

bigint operator%(const bigint &v) const { return divmod(*
    this, v).second; }

bigint &operator/=(int v) {
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = (int)z.size() - 1, rem = 0; i >= 0; --i) {
        long long cur = z[i] + rem * (long long)base;
        z[i] = (int)(cur / v);
        rem = (int)(cur % v);
    }
    trim();
    return *this;
}

bigint operator/(int v) const { return bigint(*this) /= v;
    }

int operator%(int v) const {
    if (v < 0)
        v = -v;
    int m = 0;
    for (int i = (int)z.size() - 1; i >= 0; --i)
        m = (int)((z[i] + m * (long long)base) % v);
    return m * sign;
}

bigint &operator*=(const bigint &v) {
    *this = *this * v;
    return *this;
}

bigint &operator/=(const bigint &v) {
    *this = *this / v;
    return *this;
}

bigint &operator%=(const bigint &v) {
    *this = *this % v;
    return *this;
}

bool operator<(const bigint &v) const {
    if (sign != v.sign)
        return sign < v.sign;
    if (z.size() != v.z.size())
        return z.size() * sign < v.z.size() * v.sign;
    for (int i = (int)z.size() - 1; i >= 0; i--)
        if (z[i] != v.z[i])
            return z[i] * sign < v.z[i] * sign;
    return false;
}

bool operator>(const bigint &v) const { return v < *this; }

bool operator<=(const bigint &v) const { return !(v < *this
    ); }

bool operator>=(const bigint &v) const { return !(*this < v
    ); }

bool operator==(const bigint &v) const { return sign == v.
    sign && z == v.z; }

bool operator!=(const bigint &v) const { return !(*this ==
    v); }

void trim() {
    while (!z.empty() && z.back() == 0)
        z.pop_back();
    if (z.empty())
        sign = 1;
}

bool isZero() const { return z.empty(); }

friend bigint operator-(bigint v) {
    if (!v.z.empty())
        v.sign = -v.sign;
    return v;
}

bigint abs() const { return sign == 1 ? *this : -*this; }

long long longValue() const {
    long long res = 0;
    for (int i = (int)z.size() - 1; i >= 0; i--)
        res = res * base + z[i];
    return res * sign;
}

friend bigint gcd(const bigint &a, const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b); }

friend bigint lcm(const bigint &a, const bigint &b) {
    return a / gcd(a, b) * b; }

void read(const string &s) {
    sign = 1;
    z.clear();
    int pos = 0;
    while (pos < s.size() && (s[pos] == '-' || s[pos] == '+
        ')) {
        if (s[pos] == '-')
            sign = -sign;
        ++pos;
    }
    for (int i = (int)s.size() - 1; i >= pos; i -=
        base_digits) {
        int x = 0;
        for (int j = max(pos, i - base_digits + 1); j <= i;
            j++)
            x = x * 10 + s[j] - '0';
        z.push_back(x);
    }
    trim();
}

friend istream &operator>>(istream &stream, bigint &v) {
    string s;
    stream >> s;
    v.read(s);
    return stream;
}

friend ostream &operator<<(ostream &stream, const bigint &v
    ) {
    if (v.sign == -1)
        stream << '-';
    stream << (v.z.empty() ? 0 : v.z.back());
    for (int i = (int)v.z.size() - 2; i >= 0; --i)
        stream << setw(base_digits) << setfill('0') << v.z[
            i];
    return stream;
}

static vector<int> convert_base(const vector<int> &a, int
    old_digits, int new_digits) {
    vector<long long> p(max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < p.size(); i++)
        p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int v : a) {
        cur += v * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back(int(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int)cur);
    while (!res.empty() && res.back() == 0)
        res.pop_back();
    return res;
}

bigint operator*(const bigint &v) const {
    if (min(z.size(), v.z.size()) < 150)
        return mul_simple(v);
    bigint res;
    res.sign = sign * v.sign;
    res.z = multiply_bigint(convert_base(z, base_digits,
        fft_base_digits),
                            convert_base(v.z, base_digits,
                                fft_base_digits), fft_base
                                );
    res.z = convert_base(res.z, fft_base_digits,
        base_digits);
    res.trim();
    return res;
}

bigint mul_simple(const bigint &v) const {
    bigint res;
    res.sign = sign * v.sign;
    res.z.resize(z.size() + v.z.size());
    for (int i = 0; i < z.size(); ++i)
        if (z[i])
```

```
                for (int j = 0, carry = 0; j < v.z.size() ||
                     carry; ++j) {
                    long long cur = res.z[i + j] + (long long)z
                         [i] * (j < v.z.size() ? v.z[j] : 0) +
                         carry;
                    carry = (int)(cur / base);
                    res.z[i + j] = (int)(cur % base);
                }
        res.trim();
        return res;
    }
};

mt19937 rng(1);

bigint random_bigint(int n) {
    string s;
    for (int i = 0; i < n; i++) {
        s += uniform_int_distribution<int>('0', '9')(rng);
    }
    return bigint(s);
}

// random tests
int main() {
    bigint x = bigint("120");
    bigint y = bigint("5");
    cout << x / y << endl;

    for (int i = 0; i < 1000; i++) {
        int n = uniform_int_distribution<int>(1, 100)(rng);
        bigint a = random_bigint(n);
        bigint res = sqrt(a);
        bigint xx = res * res;
        bigint yy = (res + 1) * (res + 1);

        if (xx > a || yy <= a) {
            cout << i << endl;
            cout << a << " " << res << endl;
            break;
        }

        int m = uniform_int_distribution<int>(1, n)(rng);
        bigint b = random_bigint(m) + 1;
        res = a / b;
        xx = res * b;
        yy = b * (res + 1);

        if (xx > a || yy <= a) {
            cout << i << endl;
            cout << a << " " << b << " " << res << endl;
            break;
        }
    }

    {
        bigint a = random_bigint(10'000);
        bigint b = random_bigint(2000);
        auto t1 = chrono::high_resolution_clock::now();
        bigint c = a / b;
        auto t2 = chrono::high_resolution_clock::now();
        chrono::duration<double, milli> duration = t2 - t1;
        cout << duration.count() << " ms" << endl;
        cout << endl;
    }

    bigint a = random_bigint(200'000);
    bigint b = random_bigint(200'000);
    bigint c1, c2;
    {
        auto t1 = chrono::high_resolution_clock::now();
        c1 = a * b;
        auto t2 = chrono::high_resolution_clock::now();
        chrono::duration<double, milli> duration = t2 - t1;
        cout << duration.count() << " ms" << endl;
    }
    {
        auto t1 = chrono::high_resolution_clock::now();
        c2 = a.mul_simple(b);
        auto t2 = chrono::high_resolution_clock::now();
        chrono::duration<double, milli> duration = t2 - t1;
        cout << duration.count() << " ms" << endl;
    }
    cout << (c1 == c2) << endl;
}
```

Fft.cpp

Description: desc

"fft.h" 5777cc, 9 lines

```
// usage example
int main() {
    vector<int> a{9, 9};
    vector<int> b{8, 9};
    vector<int> mul = multiply_bigint(a, b, 10);
    for (int x : mul)
        cout << x << " ";
    cout << endl;
}
```

Fft.h

Description: desc

2f0e77, 116 lines

```
// Fast Fourier transform
// https://cp-algorithms.com/algebra/fft.html
// https://drive.google.com/file/d/1
     B9BIfATnLqL6rYiE5hY9bh20SMVmHZ7/view

using cpx = complex<double>;
const double PI = acos(-1);
vector<cpx> roots = {{0, 0}, {1, 0}};

void ensure_capacity(int min_capacity) {
    for (int len = roots.size(); len < min_capacity; len *= 2)
         {
        for (int i = len >> 1; i < len; i++) {
            roots.emplace_back(roots[i]);
            double angle = 2 * PI * (2 * i + 1 - len) / (len *
                 2);
            roots.emplace_back(cos(angle), sin(angle));
        }
    }
}

void fft(vector<cpx> &z, bool inverse) {
    int n = z.size();
    assert((n & (n - 1)) == 0);
    ensure_capacity(n);
    for (unsigned i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j >= bit; bit >>= 1)
            j -= bit;
        j += bit;
        if (i < j)
            swap(z[i], z[j]);
    }
    for (int len = 1; len < n; len <<= 1) {
        for (int i = 0; i < n; i += len * 2) {
            for (int j = 0; j < len; j++) {
                cpx root = inverse ? conj(roots[j + len]) :
                     roots[j + len];
                cpx u = z[i + j];
                cpx v = z[i + j + len] * root;
                z[i + j] = u + v;
                z[i + j + len] = u - v;
            }
        }
    }
    if (inverse)
        for (int i = 0; i < n; i++)
            z[i] /= n;
}

vector<int> multiply_bigint(const vector<int> &a, const vector<
     int> &b, int base) {
    int need = a.size() + b.size();
    int n = 1;
    while (n < need)
        n <<= 1;
    vector<cpx> p(n);
    for (size_t i = 0; i < n; i++) {
        p[i] = cpx(i < a.size() ? a[i] : 0, i < b.size() ? b[i]
             : 0);
    }
    fft(p, false);
    // a[w[k]] = (p[w[k]] + conj(p[w[n-k]])) / 2
    // b[w[k]] = (p[w[k]] - conj(p[w[n-k]])) / (2*i)
    vector<cpx> ab(n);
    cpx r(0, -0.25);
    for (int i = 0; i < n; i++) {
        int j = (n - i) & (n - 1);
        ab[i] = (p[i] * p[i] - conj(p[j] * p[j])) * r;
    }
    fft(ab, true);
    vector<int> result(need);
    long long carry = 0;
    for (int i = 0; i < need; i++) {
        long long d = (long long)(ab[i].real() + 0.5) + carry;
        carry = d / base;
        result[i] = d % base;
    }
    return result;
}

vector<int> multiply_mod(const vector<int> &a, const vector<int
     > &b, int m) {
    int need = a.size() + b.size() - 1;
    int n = 1;
    while (n < need)
        n <<= 1;
    vector<cpx> A(n);
    for (size_t i = 0; i < a.size(); i++) {
        int x = (a[i] % m + m) % m;
        A[i] = cpx(x & ((1 << 15) - 1), x >> 15);
    }
    fft(A, false);

    vector<cpx> B(n);
    for (size_t i = 0; i < b.size(); i++) {
        int x = (b[i] % m + m) % m;
        B[i] = cpx(x & ((1 << 15) - 1), x >> 15);
    }
    fft(B, false);

    vector<cpx> fa(n);
    vector<cpx> fb(n);
    for (int i = 0, j = 0; i < n; i++, j = n - i) {
        cpx a1 = (A[i] + conj(A[j])) * cpx(0.5, 0);
```

```
        cpx a2 = (A[i] - conj(A[j])) * cpx(0, -0.5);
        cpx b1 = (B[i] + conj(B[j])) * cpx(0.5, 0);
        cpx b2 = (B[i] - conj(B[j])) * cpx(0, -0.5);
        fa[i] = a1 * b1 + a2 * b2 * cpx(0, 1);
        fb[i] = a1 * b2 + a2 * b1;
    }

    fft(fa, true);
    fft(fb, true);
    vector<int> res(need);
    for (int i = 0; i < need; i++) {
        long long aa = (long long)(fa[i].real() + 0.5);
        long long bb = (long long)(fb[i].real() + 0.5);
        long long cc = (long long)(fa[i].imag() + 0.5);
        res[i] = (aa % m + (bb % m << 15) + (cc % m << 30)) % m
            ;
    }
    return res;
}
```

FftSlow.cpp

Description: desc

f1b9c4, 71 lines

```
#pragma GCC optimize("Ofast")

// Fast Fourier transform, simple implementation
// https://cp-algorithms.com/algebra/fft.html

using cpx = complex<double>;
const double PI = acos(-1);

void fft(vector<cpx> &z, bool inverse) {
    size_t n = z.size();
    assert((n & (n - 1)) == 0);
    if (n == 1)
        return;
    vector<cpx> z0(n / 2);
    vector<cpx> z1(n / 2);
    for (int i = 0; i < n / 2; i++) {
        z0[i] = z[2 * i];
        z1[i] = z[2 * i + 1];
    }
    fft(z0, inverse);
    fft(z1, inverse);
    for (int i = 0; i < n / 2; ++i) {
        double ang = 2 * PI * i / n * (inverse ? -1 : 1);
        cpx w(cos(ang), sin(ang));
        z[i] = z0[i] + w * z1[i];
        z[i + n / 2] = z0[i] - w * z1[i];
        if (inverse) {
            z[i] /= 2;
            z[i + n / 2] /= 2;
        }
    }
}

vector<int> multiply_bigint(const vector<int> &a, const vector<
    int> &b) {
    int need = a.size() + b.size();
    int n = 1;
    while (n < need)
        n <<= 1;
    vector<cpx> fa(a.begin(), a.end());
    vector<cpx> fb(b.begin(), b.end());
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++) {
        fa[i] *= fb[i];
    }
    fft(fa, true);
    vector<int> result(need);
    for (int i = 0, carry = 0; i < need; i++) {
        result[i] = (int)(fa[i].real() + 0.5) + carry;
        carry = result[i] / 10;
        result[i] %= 10;
    }
    return result;
}

// usage example
int main() {
    vector<int> a{5, 1};
    vector<int> b{2, 1};
    vector<int> res = multiply_bigint(a, b);

    for (int x : res)
        cout << x << " ";
    cout << endl;
}
```

Polynom-roots.cpp

Description: desc

ee25b3, 88 lines

```
// https://en.wikipedia.org/wiki/Laguerre%27s_method

typedef complex<double> cdouble;
typedef vector<cdouble> poly;

pair<poly, cdouble> horner(const poly &a, cdouble x0) {
    int n = a.size();
    poly b = poly(max(1, n - 1));

    for (int i = n - 1; i > 0; i--)
        b[i - 1] = a[i] + (i < n - 1 ? b[i] * x0 : 0);
    return {b, a[0] + b[0] * x0};
}

cdouble eval(const poly &p, cdouble x) {
    return horner(p, x).second;
}

poly derivative(const poly &p) {
    int n = p.size();
    poly r = poly(max(1, n - 1));
    for (int i = 1; i < n; i++)
        r[i - 1] = p[i] * cdouble(i);
    return r;
}

const double EPS = 1e-9;

int cmp(cdouble x, cdouble y) {
    double diff = abs(x) - abs(y);
    return diff < -EPS ? -1 : (diff > EPS ? 1 : 0);
}

cdouble find_one_root(const poly &p0, cdouble x) {
    int n = p0.size() - 1;
    poly p1 = derivative(p0);
    poly p2 = derivative(p1);
    for (int step = 0; step < 10'000; step++) {
        cdouble y0 = eval(p0, x);
        if (cmp(y0, 0) == 0)
            break;
        cdouble G = eval(p1, x) / y0;
        cdouble H = G * G - eval(p2, x) / y0;
        cdouble R = sqrt(cdouble(n - 1) * (H * cdouble(n) - G *
            G));
        cdouble D1 = G + R;
        cdouble D2 = G - R;
        cdouble a = cdouble(n) / (cmp(D1, D2) > 0 ? D1 : D2);
        x -= a;
        if (cmp(a, 0) == 0)
            break;
    }
    return x;
}

mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
uniform_real_distribution<double> uniform(0, 1);

vector<cdouble> find_all_roots(const poly &p) {
    vector<cdouble> res;
    poly q = p;

    while (q.size() > 2) {
        cdouble z(uniform(rng), uniform(rng));
        z = find_one_root(q, z);
        z = find_one_root(p, z);
        q = horner(q, z).first;
        res.push_back(z);
    }
    res.push_back(-q[0] / q[1]);
    return res;
}

int main(int argc, char *argv[]) {
    // x^3 - 8x^2 - 13x + 140 = (x+4)(x-5)(x-7)
    poly p = {140, -13, -8, 1};

    vector<cdouble> roots = find_all_roots(p);

    for (size_t i = 0; i < roots.size(); i++) {
        if (abs(roots[i].real()) < EPS)
            roots[i] -= cdouble(roots[i].real(), 0);
        if (abs(roots[i].imag()) < EPS)
            roots[i] -= cdouble(0, roots[i].imag());
        cout << setprecision(3) << roots[i] << endl;
    }

    return 0;
}
```

parsing (6)

ExpressionParserShuntingYard.cpp

Description: desc

dd89fe, 63 lines

```
stack<int> values;
stack<char> ops;

void process_op() {
    int r = values.top();
    values.pop();
    int l = values.top();
    values.pop();
    char op = ops.top();
    ops.pop();
    switch (op) {
        case '+':
            values.push(l + r);
            break;
        case '-':
```

```
            values.push(l - r);
            break;
        case '*':
            values.push(l * r);
            break;
        case '/':
            values.push(l / r);
            break;
    }
}

int priority(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
    }
    return 0;
}

int eval(string s) {
    s = '(' + s + ')';
    for (char c : s)
        if ('0' <= c && c <= '9')
            values.push(c - '0');
        else if (c == '(')
            ops.push(c);
        else if (c == ')') {
            while (ops.top() != '(')
                process_op();
            ops.pop();
        } else {
            while (!ops.empty() && priority(ops.top()) >=
                 priority(c))
                process_op();
            ops.push(c);
        }
    return values.top();
}

// usage example
int main() {
    cout << eval("2*2+2") << endl;
    cout << eval("2+2*2") << endl;
    cout << eval("(3+2)*2") << endl;
}
```

sort (7)

Sort.cpp

Description: desc

4a3cd0, 48 lines

```
void merge_sort(vector<int> &a, int low, int high) {
    if (high - low < 2)
        return;
    int mid = (low + high) >> 1;
    merge_sort(a, low, mid);
    merge_sort(a, mid, high);
    vector<int> b;
    copy(a.begin() + low, a.begin() + mid, back_inserter(b));
    for (int i = low, j = mid, k = 0; k < b.size(); i++) {
        if (j == high || b[k] <= a[j]) {
            a[i] = b[k++];
        } else {
            a[i] = a[j++];
        }
    }
}

void counting_sort(vector<int> &a) {
    int max = *max_element(a.begin(), a.end());
    vector<int> cnt(max + 1);
    for (int x : a) {
        ++cnt[x];
    }
    for (int i = 1; i < cnt.size(); i++) {
        cnt[i] += cnt[i - 1];
    }
    int n = a.size();
    vector<int> b(n);
    for (int i = 0; i < n; i++) {
        b[--cnt[a[i]]] = a[i];
    }
    a = b;
}

// usage example
int main() {
    vector<int> a{4, 1, 2, 3};
    merge_sort(a, 0, a.size());
    for (int x : a)
        cout << x << " ";
    cout << endl;

    a = {4, 1, 2, 3};
    counting_sort(a);
    for (int x : a)
        cout << x << " ";
    cout << endl;
}
```

strings (8)

Aho-corasick.cpp

Description: desc

b06f55, 55 lines

```
constexpr int ALPHABET_SIZE = 26;
constexpr int MAX_STATES = 200'000;

int transitions[MAX_STATES][ALPHABET_SIZE];
int sufflink[MAX_STATES];
int escape[MAX_STATES];
int states = 1;

int addString(const string &s) {
    int v = 0;
    for (char c : s) {
        c -= 'a';
        if (transitions[v][c] == 0) {
            transitions[v][c] = states++;
        }
        v = transitions[v][c];
    }
    escape[v] = v;
    return v;
}

void buildLinks() {
    vector<int> q(MAX_STATES);
    for (int s = 0, t = 1; s < t;) {
        int v = q[s++];
        int u = sufflink[v];
        if (escape[v] == 0) {
            escape[v] = escape[u];
        }
        for (int c = 0; c < ALPHABET_SIZE; c++) {
            if (transitions[v][c] != 0) {
                q[t++] = transitions[v][c];
                sufflink[transitions[v][c]] = v != 0 ?
                     transitions[u][c] : 0;
            } else {
                transitions[v][c] = transitions[u][c];
            }
        }
    }
}

int main() {
    addString("a");
    addString("aa");
    addString("abaaa");
    buildLinks();

    string s = "abaa";

    int state = 0;
    for (size_t i = 0; i < s.length(); i++) {
        state = transitions[state][s[i] - 'a'];
        if (escape[state] != 0)
            cout << i << endl;
    }
}
```

Hashing.cpp

Description: desc

6b1f33, 53 lines

```
// see https://codeforces.com/blog/entry/60442 for analysis

struct hashing {
    static constexpr int dimensions = 4;
    static constexpr int mod = (1u << 31) - 1;
    vector<vector<int>> hashes, p;

    static const vector<int> &get_bases() {
        static mt19937 rng(chrono::steady_clock::now().
             time_since_epoch().count());
        static vector<int> bases;
        while (bases.size() < dimensions) {
            bases.emplace_back(uniform_int_distribution<int>((
                 int)1e9, mod - 1)(rng));
        }
        return bases;
    }

    hashing(const string &s) : hashes(dimensions), p(dimensions
         ) {
        int n = s.size();
        const vector<int> &bases = get_bases();
        for (int d = 0; d < dimensions; ++d) {
            hashes[d].resize(n + 1);
            p[d].resize(n + 1);
            p[d][0] = 1;
            long long base = bases[d];
            for (int i = 0; i < n; i++) {
                hashes[d][i + 1] = (hashes[d][i] * base + s[i])
                     % mod;
                p[d][i + 1] = p[d][i] * base % mod;
            }
        }
    }

    vector<int> get_hash(int i, int len) {
```

```
        vector<int> res;
        for (int d = 0; d < dimensions; ++d) {
            int hash = (int)((hashes[d][i + len] + (long long)
                hashes[d][i] * (mod - p[d][len])) % mod);
            res.emplace_back(hash);
        }
        return res;
    }
};

// usage example
int main() {
    string a = "abc123";
    string b = "abc";
    auto ha = hashing(a);
    auto hb = hashing(b);
    vector<int> ha1 = ha.get_hash(0, 3);
    vector<int> ha2 = ha.get_hash(3, 3);
    vector<int> hb1 = ha.get_hash(0, 3);
    cout << (ha1 == hb1) << " " << (ha1 == ha2) << endl;
    cout << hashing::mod << endl;
}
```

Kmp.cpp

Description: Knuth-Morris-Pratt String Matching

02d85e, 34 lines

```
vector<int> prefix_function(string s) {
    vector<int> p(s.length());
    int k = 0;
    for (int i = 1; i < s.length(); i++) {
        while (k > 0 && s[k] != s[i])
            k = p[k - 1];
        if (s[k] == s[i])
            ++k;
        p[i] = k;
    }
    return p;
}

int find_substring(string haystack, string needle) {
    int m = needle.length();
    if (m == 0)
        return 0;
    vector<int> p = prefix_function(needle);
    for (int i = 0, k = 0; i < haystack.length(); i++) {
        while (k > 0 && needle[k] != haystack[i])
            k = p[k - 1];
        if (needle[k] == haystack[i])
            ++k;
        if (k == m)
            return i + 1 - m;
    }
    return -1;
}

// usage example
int main() {
    int pos = find_substring("acabc", "ab");
    cout << pos << endl;
}
```

Manacher.cpp

Description: desc

39033f, 52 lines

```
// Manacher's algorithm: https://cp-algorithms.com/string/
    manacher.html

// d1[i] - how many palindromes of odd length with center at i
vector<int> odd_palindromes(const string &s) {
    size_t n = s.size();
    vector<int> d1(n);
    int l = 0, r = -1;
    for (int i = 0; i < n; ++i) {
        int len = i > r ? 1 : min(d1[l + r - i], r - i + 1);
        while (i - len >= 0 && i + len < n && s[i - len] == s[i
            + len])
            ++len;
        d1[i] = len;
        if (r < i + len - 1) {
            r = i + len - 1;
            l = i - (len - 1);
        }
    }
    return d1;
}

// d2[i] - how many palindromes of even length with center at i
vector<int> even_palindromes(const string &s) {
    size_t n = s.size();
    vector<int> d2(n);
    int l = 0, r = -1;
    for (int i = 0; i < n; ++i) {
        int len = i > r ? 0 : min(d2[l + r - i + 1], r - i + 1)
            ;
        while (i - len - 1 >= 0 && i + len < n && s[i - len -
            1] == s[i + len])
            ++len;
        d2[i] = len;
        if (r < i + len - 1) {
            r = i + len - 1;
            l = i - len;
        }
    }
    return d2;
}

// usage example
int main() {
    string text = "abbba";

    auto d1 = odd_palindromes(text);
    for (int d : d1)
        cout << d << " ";
    cout << endl;

    auto d2 = even_palindromes(text);
    for (int d : d2)
        cout << d << " ";
    cout << endl;
}
```

MinRotation.cpp

Description: desc

7b7f92, 25 lines

```
string min_cyclic_shift(string s) {
    s += s;
    int n = s.length();
    int i = 0;
    int pos = 0;
    while (i < n / 2) {
        pos = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                ++k;
            ++j;
        }
        while (i <= k)
            i += j - k;
    }
    return s.substr(pos, n / 2);
}

// usage example
int main() {
    cout << min_cyclic_shift("bcabab") << endl;
}
```

Suffix-array-sa-is.cpp

Description: Suffix array and lcp in O(n)

Time: $\mathcal{O}(n)$

6c553f, 155 lines

```
#define tget(i) ((t[(i) / 8] & (1 << ((i) % 8))) ? 1 : 0)
#define tset(i, b) t[(i) / 8] = (b) ? ((1 << ((i) % 8)) | t[(i)
    / 8]) : ((~(1 << ((i) % 8))) & t[(i) / 8])
#define chr(i) (cs == sizeof(int) ? ((int *)s)[i] : ((unsigned
    char *)s)[i])
#define isLMS(i) (i > 0 && tget(i) && !tget(i - 1))

// find the start or end of each bucket
void getBuckets(unsigned char *s, int *bkt, int n, int K, int
    cs, bool end) {
    int i, sum = 0;
    for (i = 0; i <= K; i++)
        bkt[i] = 0;  // clear all buckets
    for (i = 0; i < n; i++)
        bkt[chr(i)]++;  // compute the size of each bucket
    for (i = 0; i <= K; i++) {
        sum += bkt[i];
        bkt[i] = end ? sum : sum - bkt[i];
    }
}

// compute SAl
void induceSAl(unsigned char *t, int *SA, unsigned char *s, int
    *bkt, int n, int K, int cs, bool end) {
    int i, j;
    getBuckets(s, bkt, n, K, cs, end);  // find starts of
        buckets
    for (i = 0; i < n; i++) {
        j = SA[i] - 1;
        if (j >= 0 && !tget(j))
            SA[bkt[chr(j)]++] = j;
    }
}

// compute SAs
void induceSAs(unsigned char *t, int *SA, unsigned char *s, int
    *bkt, int n, int K, int cs, bool end) {
    int i, j;
    getBuckets(s, bkt, n, K, cs, end);  // find ends of buckets
    for (i = n - 1; i >= 0; i--) {
        j = SA[i] - 1;
        if (j >= 0 && tget(j))
            SA[--bkt[chr(j)]] = j;
    }
}

// find the suffix array SA of s[0..n-1] in {1..K}^n
// require s[n-1]=0 (the sentinel!), n>=2
// use a working space (excluding s and SA) of at most 2.25n+O
    (1) for a constant alphabet
void SA_IS(unsigned char *s, int *SA, int n, int K, int cs) {
    int i, j;
```

```
unsigned char *t = (unsigned char *)malloc(n / 8 + 1);  // LS-type array in bits
// Classify the type of each character
tset(n - 2, 0);
tset(n - 1, 1);  // the sentinel must be in s1, important !!!
for (i = n - 3; i >= 0; i--)
    tset(i, (chr(i) < chr(i + 1) || (chr(i) == chr(i + 1)
        && tget(i + 1) == 1)) ? 1 : 0);
// stage 1: reduce the problem by at least 1/2
// sort all the S-substrings
int *bkt = (int *)malloc(sizeof(int) * (K + 1));  // bucket array
getBuckets(s, bkt, n, K, cs, true);               // find ends of buckets
for (i = 0; i < n; i++)
    SA[i] = -1;
for (i = 1; i < n; i++)
    if (isLMS(i))
        SA[--bkt[chr(i)]] = i;
induceSA1(t, SA, s, bkt, n, K, cs, false);
induceSAs(t, SA, s, bkt, n, K, cs, true);
free(bkt);
// compact all the sorted substrings into the first n1 items of SA
// 2*n1 must be not larger than n (proveable)
int n1 = 0;
for (i = 0; i < n; i++)
    if (isLMS(SA[i]))
        SA[n1++] = SA[i];
// find the lexicographic names of all substrings
for (i = n1; i < n; i++)
    SA[i] = -1;  // init the name array buffer
int name = 0, prev = -1;
for (i = 0; i < n1; i++) {
    int pos = SA[i];
    bool diff = false;
    for (int d = 0; d < n; d++)
        if (prev == -1 || chr(pos + d) != chr(prev + d) ||
            tget(pos + d) != tget(prev + d)) {
            diff = true;
            break;
        } else if (d > 0 && (isLMS(pos + d) || isLMS(prev + d)))
            break;
    if (diff) {
        name++;
        prev = pos;
    }
    pos = (pos % 2 == 0) ? pos / 2 : (pos - 1) / 2;
    SA[n1 + pos] = name - 1;
}
for (i = n - 1, j = n - 1; i >= n1; i--)
    if (SA[i] >= 0)
        SA[j--] = SA[i];
// stage 2: solve the reduced problem
// recurse if names are not yet unique
int *SA1 = SA, *s1 = SA + n - n1;
if (name < n1)
    SA_IS((unsigned char *)s1, SA1, n1, name - 1, sizeof(int));
else
    // generate the suffix array of s1 directly
    for (i = 0; i < n1; i++)
        SA1[s1[i]] = i;
// stage 3: induce the result for the original problem
bkt = (int *)malloc(sizeof(int) * (K + 1));  // bucket array
// put all left-most S characters into their buckets
    getBuckets(s, bkt, n, K, cs, true);  // find ends of buckets
    for (i = 1, j = 0; i < n; i++)
        if (isLMS(i))
            s1[j++] = i;  // get p1
    for (i = 0; i < n1; i++)
        SA1[i] = s1[SA1[i]];  // get index in s
    for (i = n1; i < n; i++)
        SA[i] = -1;  // init SA[n1..n-1]
    for (i = n1 - 1; i >= 0; i--) {
        j = SA[i];
        SA[i] = -1;
        SA[--bkt[chr(j)]] = j;
    }
    induceSA1(t, SA, s, bkt, n, K, cs, false);
    induceSAs(t, SA, s, bkt, n, K, cs, true);
    free(bkt);
    free(t);
}

vector<int> suffix_array(const string &s) {
    int n = s.size();
    if (n == 0)
        return {};
    if (n == 1)
        return {0};
    vector<int> sa(n + 1);
    SA_IS((unsigned char *)s.c_str(), sa.data(), n + 1, 256, 1);
    sa.erase(sa.begin());
    return sa;
}

// usage example
int main() {
    string s = "abcab";
    vector<int> sa = suffix_array(s);
    for (int v : sa)
        cout << v << " ";
    cout << endl;

    mt19937 rng(1);
    s.clear();
    for (int i = 0; i < 1000'000; ++i) {
        char c = uniform_int_distribution<int>('a', 'd')(rng);
        s.push_back(c);
    }
    auto t1 = chrono::high_resolution_clock::now();
    sa = suffix_array(s);
    auto t2 = chrono::high_resolution_clock::now();
    chrono::duration<double, milli> duration = t2 - t1;
    cout << duration.count() << " ms" << endl;
}
```

Suffix-array.cpp

Description: Build suffix array
Time: $\mathcal{O}(nlogn)$

1dc346, 93 lines

```
vector<int> suffix_array(const string &S) {
    int n = S.length();

    // Stable sort of characters.
    // Same characters are sorted by their position in descending order.
    // E.g. last character which represents suffix of length 1 should be ordered first among same characters.
    vector<int> sa;
    for (int i = n - 1; i >= 0; --i) {
        sa.push_back(i);
    }
    stable_sort(sa.begin(), sa.end(), [&](int a, int b) {
        return S[a] < S[b]; });

    vector<int> classes(n);
    for (int i = 0; i < n; ++i) {
        classes[i] = S[i];
    }
    // sa[i] - suffix on i'th position after sorting by first len characters
    // classes[i] - equivalence class of the i'th suffix after sorting by first len characters

    for (int len = 1; len < n; len *= 2) {
        // Calculate classes for suffixes of length len * 2
        vector<int> c = classes;
        for (int i = 0; i < n; i++) {
            // Condition sa[i - 1] + len < n emulates 0-symbol at the end of the string.
            // A separate class is created for each suffix followed by emulated 0-symbol.
            classes[sa[i]] =
                i > 0 && c[sa[i - 1]] == c[sa[i]] && sa[i - 1] + len < n && c[sa[i - 1] + len / 2] == c[sa[i] + len / 2]
                ? classes[sa[i - 1]]
                : i;
        }
        // Suffixes are already sorted by first len characters
        // Now sort suffixes by first len * 2 characters
        vector<int> cnt(n);
        iota(cnt.begin(), cnt.end(), 0);
        vector<int> s = sa;
        for (int i = 0; i < n; i++) {
            // s[i] - order of suffixes sorted by first len characters
            // (s[i] - len) - order of suffixes sorted only by second len characters
            int s1 = s[i] - len;
            // sort only suffixes of length > len, others are already sorted
            if (s1 >= 0)
                sa[cnt[classes[s1]]++] = s1;
        }
    }
    return sa;
}

// https://en.wikipedia.org/wiki/LCP_array
vector<int> lcp_array(const string &s) {
    int n = s.size();
    vector<int> sa = suffix_array(s);
    vector<int> rank(n);
    for (int i = 0; i < n; i++)
        rank[sa[i]] = i;
    vector<int> lcp(n - 1);
    for (int i = 0, h = 0; i < n; i++) {
        if (rank[i] < n - 1) {
            for (int j = sa[rank[i] + 1]; s[i + h] == s[j + h]; ++h)
                ;
            lcp[rank[i]] = h;
            if (h > 0)
                --h;
        }
    }
    return lcp;
}
```

```
// usage example
int main() {
    string s = "abcab";

    vector<int> sa = suffix_array(s);
    for (int v : sa)
        cout << v << " ";
    cout << endl;

    vector<int> lcp = lcp_array(s);
    for (int v : lcp)
        cout << v << " ";
    cout << endl;

    mt19937 rng(1);
    s.clear();
    for (int i = 0; i < 1000'000; ++i) {
        char c = uniform_int_distribution<int>('a', 'd')(rng);
        s.push_back(c);
    }
    auto t1 = chrono::high_resolution_clock::now();
    sa = suffix_array(s);
    auto t2 = chrono::high_resolution_clock::now();
    chrono::duration<double, milli> duration = t2 - t1;
    cout << duration.count() << " ms" << endl;
}
```

Suffix-automaton.cpp

Description: desc

f7d483, 56 lines

```
// https://cp-algorithms.com/string/suffix-automaton.html

struct state {
    int length;
    int suffLink;
    vector<int> inv_sufflinks;
    int firstPos = -1;
    vector<int> next = vector<int>(128, -1);
};

constexpr int MAXLEN = 100'000;
state st[MAXLEN * 2];
int sz;

void build_suffix_automaton(const string &s) {
    st[0].suffLink = -1;
    int last = 0;
    sz = 1;
    for (int i = 0; i < s.length(); i++) {
        char c = s[i];
        int cur = sz++;
        st[cur].length = i + 1;
        st[cur].firstPos = i;
        int p = last;
        for (; p != -1 && st[p].next[c] == -1; p = st[p].
             suffLink) {
            st[p].next[c] = cur;
        }
        if (p == -1) {
            st[cur].suffLink = 0;
        } else {
            int q = st[p].next[c];
            if (st[p].length + 1 == st[q].length) {
                st[cur].suffLink = q;
            } else {
                int clone = sz++;
                st[clone].length = st[p].length + 1;
                st[clone].next = st[q].next;
                st[clone].suffLink = st[q].suffLink;
                for (; p != -1 && st[p].next[c] == q; p = st[p
                     ].suffLink) {
                    st[p].next[c] = clone;
                }
                st[q].suffLink = clone;
                st[cur].suffLink = clone;
            }
        }
        last = cur;
    }
    for (int i = 1; i < sz; i++) {
        st[st[i].suffLink].inv_sufflinks.push_back(i);
    }
}

// usage example
int main() {
    build_suffix_automaton("ababcc");
}
```

SuffixTreeBreslauerItaliano.cpp

Description: desc

ce7743, 49 lines

```
// See https://codeforces.com/blog/entry/17956?#comment-228040
     for description

const int MAXLEN = 1e6;
string s;
int pos[MAXLEN], len[MAXLEN], par[MAXLEN];
unordered_map<char, int> to[MAXLEN], Link[MAXLEN];
int sz = 2;

void attach(int child, int parent, char c, int child_len) {
    to[parent][c] = child;
    len[child] = child_len;
    par[child] = parent;
}

void extend(char c) {
    int v;
    int i = s.size();
    int old = sz - 1;
    for (v = old; !Link[v].count(c); v = par[v])
        i -= len[v];
    int w = Link[v][c];
    if (to[w].count(s[i])) {
        int u = to[w][c];
        for (pos[sz] = pos[u] - len[u]; s[pos[sz]] == s[i]; pos
             [sz] += len[v], i += len[v])
            v = to[v][s[i]];
        attach(sz, w, s[pos[u] - len[u]], len[u] - (pos[u] -
             pos[sz]));
        attach(u, sz, s[pos[sz]], pos[u] - pos[sz]);
        w = Link[v][c] = sz++;
    }
    Link[old][c] = sz;
    attach(sz, w, s[i], s.size() - i);
    pos[sz++] = s.size();
}

void init_tree() {
    len[1] = 1;
    pos[1] = 0;
    par[1] = 0;
    for (int c = 0; c <= 255; c++)
        to[0][c] = Link[0][c] = 1;
}

int main() {
    init_tree();

    s = "aababababbbadcasdf#";
    for (int i = s.size() - 1; i >= 0; i--)
        extend(s[i]);
}
```

SuffixTreeUkkonen.cpp

Description: desc

8e6bc3, 66 lines

```
// See http://codeforces.ru/blog/entry/16780 for description

const int inf = 1e9;
const int maxn = 1e5;
char s[maxn];
unordered_map<char, int> to[maxn];
int len[maxn], f_pos[maxn], Link[maxn];
int node, pos;
int sz = 1, n = 0;

int make_node(int _pos, int _len) {
    f_pos[sz] = _pos;
    len[sz] = _len;
    return sz++;
}

void go_edge() {
    while (pos > len[to[node][s[n - pos]]]) {
        node = to[node][s[n - pos]];
        pos -= len[node];
    }
}

void add_letter(char c) {
    s[n++] = c;
    pos++;
    int last = 0;
    while (pos > 0) {
        go_edge();
        int edge = s[n - pos];
        int &v = to[node][edge];
        int t = s[f_pos[v] + pos - 1];
        if (v == 0) {
            v = make_node(n - pos, inf);
            Link[last] = node;
            last = 0;
        } else if (t == c) {
            Link[last] = node;
            return;
        } else {
            int u = make_node(f_pos[v], pos - 1);
            to[u][c] = make_node(n - 1, inf);
            to[u][t] = v;
            f_pos[v] += pos - 1;
            len[v] -= pos - 1;
            v = u;
            Link[last] = u;
            last = u;
        }
        if (node == 0)
            pos--;
        else
            node = Link[node];
    }
}

int main() {
    len[0] = inf;
    string s = "abracadabra";
```

```
    int ans = 0;
    for (int i = 0; i < s.size(); i++)
        add_letter(s[i]);
    for (int i = 1; i < sz; i++)
        ans += min((int)s.size() - f_pos[i], len[i]);
    cout << ans << "\n";
}
```

TandemRepeats.cpp

Description: desc

ae84b2, 75 lines

```
vector<int> z_function(const string &s) {
    int n = (int)s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}

void output_tandem(const string &s, int shift, bool left, int
     cntr, int l, int l1, int l2) {
    int pos;
    if (left)
        pos = cntr - l1;
    else
        pos = cntr - l1 - l2 - l1 + 1;
    cout << "[" << shift + pos << ".." << shift + pos + 2 * l -
          1 << "] = " << s.substr(pos, 2 * l) << endl;
}

void output_tandems(const string &s, int shift, bool left, int
     cntr, int l, int k1, int k2) {
    for (int l1 = 1; l1 <= l; ++l1) {
        if (left && l1 == l)
            break;
        if (l1 <= k1 && l - l1 <= k2)
            output_tandem(s, shift, left, cntr, l, l1, l - l1);
    }
}

inline int get_z(const vector<int> &z, int i) {
    return 0 <= i && i < (int)z.size() ? z[i] : 0;
}

void find_tandems(string s, int shift = 0) {
    int n = (int)s.length();
    if (n == 1)
        return;

    int nu = n / 2;
    int nv = n - nu;
    string u = s.substr(0, nu);
    string v = s.substr(nu);
    string ru = string(u.rbegin(), u.rend());
    string rv = string(v.rbegin(), v.rend());

    find_tandems(u, shift);
    find_tandems(v, shift + nu);

    vector<int> z1 = z_function(ru);
    vector<int> z2 = z_function(v + '#' + u);
    vector<int> z3 = z_function(ru + '#' + rv);
    vector<int> z4 = z_function(v);
    for (int cntr = 0; cntr < n; ++cntr) {
        int l, k1, k2;
        if (cntr < nu) {
            l = nu - cntr;
            k1 = get_z(z1, nu - cntr);
            k2 = get_z(z2, nv + 1 + cntr);
        } else {
            l = cntr - nu + 1;
            k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
            k2 = get_z(z4, (cntr - nu) + 1);
        }
        if (k1 + k2 >= l)
            output_tandems(s, shift, cntr < nu, cntr, l, k1, k2
                );
    }
}

// usage example
int main() {
    find_tandems("abcabczz");
}
```

Z-function.cpp

Description: desc

e89fcd, 23 lines

```
// z[i] = lcp(s[0..], s[i..])
vector<int> z_function(const string &s) {
    vector<int> z(s.length());
    for (int i = 1, l = 0, r = 0; i < z.size(); ++i) {
        if (i <= r)
            z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < z.size() && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (r < i + z[i] - 1) {
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}

// usage example
int main() {
    vector<int> z = z_function("abcababc");
    for (int x : z)
        cout << x << " ";
    cout << endl;
}
```

structures (9)

BinaryHeapIndexed.cpp

Description: desc

"binary_heap_indexed.h"

823ead, 12 lines

```
// usage example
int main() {
    binary_heap_indexed<int> h(5);
    h.add(0, 50);
    h.add(1, 30);
    h.add(2, 40);
    h.change_value(0, 20);
    h.remove(1);
    while (h.size) {
        cout << h.remove_min() << endl;
    }
}
```

BinaryHeapIndexed.h

Description: desc

568d07, 76 lines

```
template <class T>
struct binary_heap_indexed {
    vector<T> heap;
    vector<int> pos2Id;
    vector<int> id2Pos;
    int size;

    binary_heap_indexed() : size(0) {}

    binary_heap_indexed(int n) : heap(n), pos2Id(n), id2Pos(n),
          size(0) {}

    void add(int id, T value) {
        heap[size] = value;
        pos2Id[size] = id;
        id2Pos[id] = size;
        up(size++);
    }

    int remove_min() {
        int removedId = pos2Id[0];
        heap[0] = heap[--size];
        pos2Id[0] = pos2Id[size];
        id2Pos[pos2Id[0]] = 0;
        down(0);
        return removedId;
    }

    void remove(int id) {
        int pos = id2Pos[id];
        pos2Id[pos] = pos2Id[--size];
        id2Pos[pos2Id[pos]] = pos;
        change_value(pos2Id[pos], heap[size]);
    }

    void change_value(int id, T value) {
        int pos = id2Pos[id];
        if (heap[pos] < value) {
            heap[pos] = value;
            down(pos);
        } else if (heap[pos] > value) {
            heap[pos] = value;
            up(pos);
        }
    }

    void up(int pos) {
        while (pos > 0) {
            int parent = (pos - 1) / 2;
            if (heap[pos] >= heap[parent])
                break;
            exchange(pos, parent);
            pos = parent;
        }
    }

    void down(int pos) {
        while (true) {
            int child = 2 * pos + 1;
            if (child >= size)
                break;
            if (child + 1 < size && heap[child + 1] < heap[
                 child])
                ++child;
            if (heap[pos] <= heap[child])
                break;
```

```
            exchange(pos, child);
            pos = child;
        }
    }

    void exchange(int i, int j) {
        swap(heap[i], heap[j]);
        swap(pos2Id[i], pos2Id[j]);
        id2Pos[pos2Id[i]] = i;
        id2Pos[pos2Id[j]] = j;
    }
};
```

CentroidDecomposition.cpp

Description: desc

efbefe, 54 lines

```
void calc_sizes(const vector<vector<int>> &tree, vector<int> &
     size, vector<bool> &deleted, int u, int p) {
    size[u] = 1;
    for (int v : tree[u]) {
        if (v == p || deleted[v])
            continue;
        calc_sizes(tree, size, deleted, v, u);
        size[u] += size[v];
    }
}

int find_tree_centroid(const vector<vector<int>> &tree, vector<
     int> &size, vector<bool> &deleted, int u, int p,
                       int vertices) {
    for (int v : tree[u]) {
        if (v == p || deleted[v])
            continue;
        if (size[v] > vertices / 2) {
            return find_tree_centroid(tree, size, deleted, v, u
                , vertices);
        }
    }
    return u;
}

// void dfs(const vector<vector<int>> &tree, vector<bool> &
     deleted, int u, int p) {
//     for (int v: tree[u]) {
//         if (v == p || deleted[v])continue;
//         dfs(tree, deleted, v, u);
//     }
// }

void decompose(const vector<vector<int>> &tree, vector<int> &
     size, vector<bool> &deleted, int u, int total) {
    calc_sizes(tree, size, deleted, u, -1);
    int centroid = find_tree_centroid(tree, size, deleted, u,
         -1, total);
    deleted[centroid] = true;

    // process centroid vertex here
    // dfs(tree, deleted, centroid, -1);
    cout << centroid << endl;

    for (int v : tree[centroid]) {
        if (deleted[v])
            continue;
        decompose(tree, size, deleted, v, size[v]);
    }
}

// usage example
int main() {
    vector<vector<int>> tree = {{3}, {3}, {3}, {0, 1, 2}};

    int n = tree.size();
    vector<int> size(n);
    vector<bool> deleted(n);
    decompose(tree, size, deleted, 0, n);
}
```

DisjointSets.cpp

Description: desc

d68061, 26 lines

```
// https://cp-algorithms.com/data_structures/disjoint_set_union
    .html

vector<int> create_sets(int n) {
    vector<int> res(n);
    iota(res.begin(), res.end(), 0);
    return res;
}

int root(vector<int> &p, int x) {
    return x == p[x] ? x : (p[x] = root(p, p[x]));
}

void unite(vector<int> &p, int a, int b) {
    a = root(p, a);
    b = root(p, b);
    p[b] = a;
}

// usage example
int main() {
    vector<int> p = create_sets(3);
    unite(p, 0, 2);
    cout << (0 == root(p, 0)) << endl;
    cout << (1 == root(p, 1)) << endl;
    cout << (0 == root(p, 2)) << endl;
}
```

DisjointSetsRanked.cpp

Description: desc

b68d21, 33 lines

```
// https://cp-algorithms.com/data_structures/disjoint_set_union
    .html

tuple<vector<int>, vector<int>> create_sets(int n) {
    vector<int> res(n);
    iota(res.begin(), res.end(), 0);
    vector<int> rank(n);
    return tuple{res, rank};
}

int root(vector<int> &p, int x) {
    return x == p[x] ? x : (p[x] = root(p, p[x]));
}

void unite(vector<int> &p, vector<int> &rank, int a, int b) {
    a = root(p, a);
    b = root(p, b);
    if (a == b)
        return;
    if (rank[a] < rank[b])
        swap(a, b);
    if (rank[a] == rank[b])
        ++rank[a];
    p[b] = a;
}

// usage example
int main() {
    auto [p, rank] = create_sets(3);
    unite(p, rank, 0, 2);
    cout << (0 == root(p, 0)) << endl;
    cout << (1 == root(p, 1)) << endl;
    cout << (0 == root(p, 2)) << endl;
}
```

FenwickTree.cpp

Description: desc

ff305c, 53 lines

```
// https://cp-algorithms.com/data_structures/fenwick.html

template <class T>
struct fenwick {
    vector<T> t;

    fenwick(int n) : t(n) {}

    void add(int i, T value) {
        for (; i < t.size(); i |= i + 1)
            t[i] += value;
    }

    // sum[0..i]
    T sum(int i) {
        T res{};
        for (; i >= 0; i = (i & (i + 1)) - 1)
            res += t[i];
        return res;
    }

    // returns min(p | sum[0..p] >= sum)
    // requires non-negative tree values
    int lower_bound(T sum) {
        int highest_one_bit = 1;
        while (highest_one_bit << 1 <= t.size())
            highest_one_bit <<= 1;
        int pos = 0;
        for (size_t blockSize = highest_one_bit; blockSize !=
             0; blockSize >>= 1) {
            int p = pos + blockSize - 1;
            if (p < t.size() && t[p] < sum) {
                sum -= t[p];
                pos += blockSize;
            }
        }
        return pos;
    }
};

// usage example
int main() {
    fenwick<int> f(3);
    f.add(0, 4);
    f.add(1, 5);
    f.add(2, 5);
    f.add(2, 5);

    cout << boolalpha;
    cout << (4 == f.sum(0)) << endl;
    cout << (19 == f.sum(2)) << endl;
    cout << (2 == f.lower_bound(19)) << endl;
    cout << (3 == f.lower_bound(20)) << endl;
}
```

FenwickTreeInterval.cpp

Description: desc

651be8, 44 lines

```
// https://cp-algorithms.com/data_structures/fenwick.html

template <class T>
class fenwick_interval {
    vector<T> t1, t2;

    void add(vector<T> &t, int i, T value) {
        for (; i < t.size(); i |= i + 1)
            t[i] += value;
    }

    T sum(vector<T> &t, int i) {
        T res = 0;
        for (; i >= 0; i = (i & (i + 1)) - 1)
            res += t[i];
        return res;
    }

   public:
    fenwick_interval(int n) : t1(n), t2(n) {}

    void add(int a, int b, T value) {
        add(t1, a, value);
        add(t1, b, -value);
        add(t2, a, -value * (a - 1));
        add(t2, b, value * b);
    }

    // sum[0..i]
    T sum(int i) { return sum(t1, i) * i + sum(t2, i); }
};

// usage example
int main() {
    fenwick_interval<int> f(3);
    f.add(0, 0, 4);
    f.add(1, 1, 5);
    f.add(2, 2, 5);
    f.add(2, 2, 5);

    cout << boolalpha;
    cout << (4 == f.sum(0)) << endl;
    cout << (19 == f.sum(2)) << endl;
}
```

FenwickTreeOnMap.cpp

Description: desc

54c78e, 28 lines

```
const int n = 2000'000'000;

void add(unordered_map<int, int> &t, int i, int value) {
    for (; i < n; i |= i + 1)
        t[i] += value;
}

// sum[0, i]
int sum(unordered_map<int, int> &t, int i) {
    int res = 0;
    for (; i >= 0; i = (i & (i + 1)) - 1)
        if (t.count(i))
            res += t[i];
    return res;
}

// Usage example
int main() {
    unordered_map<int, int> t;
    add(t, 0, 4);
    add(t, 1, 5);
    add(t, 2, 5);
    add(t, 2, 5);

    cout << (4 == sum(t, 0)) << endl;
    cout << (19 == sum(t, 2)) << endl;
    cout << (19 == sum(t, 1000'000'000)) << endl;
}
```

HeavyLightDecomposition.cpp

Description: desc

"segment_tree.h" 677f5c, 88 lines

```
class HeavyLight {
   public:
    vector<vector<int>> tree;
    bool valuesOnVertices;  // true - values on vertices, false
        - values on edges
    segtree segment_tree;
    vector<int> parent;
    vector<int> depth;
    vector<int> pathRoot;
    vector<int> in;

    HeavyLight(const vector<vector<int>> &t, bool
        valuesOnVertices)
        : tree(t),
          valuesOnVertices(valuesOnVertices),
          segment_tree(t.size()),
          parent(t.size()),
          depth(t.size()),
          pathRoot(t.size()),
          in(t.size()) {
        int time = 0;
        parent[0] = -1;

        function<int(int)> dfs1 = [&](int u) {
            int size = 1;
            int maxSubtree = 0;
            for (int &v : tree[u]) {
                if (v == parent[u])
                    continue;
                parent[v] = u;
                depth[v] = depth[u] + 1;
                int subtree = dfs1(v);
                if (maxSubtree < subtree) {
                    maxSubtree = subtree;
                    swap(v, tree[u][0]);
                }
                size += subtree;
            }
            return size;
        };

        function<void(int)> dfs2 = [&](int u) {
            in[u] = time++;
            for (int v : t[u]) {
                if (v == parent[u])
                    continue;
                pathRoot[v] = v == t[u][0] ? pathRoot[u] : v;
                dfs2(v);
            }
        };

        dfs1(0);
        dfs2(0);
    }

    segtree::node get(int u, int v) {
        segtree::node res;
        process_path(u, v, [this, &res](int a, int b) { res =
            segtree::unite(res, segment_tree.get(a, b)); });
        return res;
    }

    void modify(int u, int v, long long delta) {
        process_path(u, v, [this, delta](int a, int b) {
            segment_tree.modify(a, b, delta); });
    }

    void process_path(int u, int v, const function<void(int x,
        int y)> &op) {
        for (; pathRoot[u] != pathRoot[v]; v = parent[pathRoot[
            v]]) {
            if (depth[pathRoot[u]] > depth[pathRoot[v]])
                swap(u, v);
            op(in[pathRoot[v]], in[v]);
        }
        if (u != v || valuesOnVertices)
            op(min(in[u], in[v]) + (valuesOnVertices ? 0 : 1),
                max(in[u], in[v]));
    }
};

// usage example
int main() {
    vector<vector<int>> tree{{1, 2}, {0, 3, 4}, {0}, {1}, {1}};

    HeavyLight hl_v(tree, true);
    hl_v.modify(3, 2, 1);
    hl_v.modify(1, 0, -1);
    cout << hl_v.get(4, 2).sum << endl;

    HeavyLight hl_e(tree, false);
    hl_e.modify(3, 2, 1);
    hl_e.modify(1, 0, -1);
    cout << hl_e.get(4, 2).sum << endl;
}
```

KdTree.cpp

Description: desc

853a9d, 91 lines

```
using pii = pair<int, int>;

const int maxn = 100'000;
int tx[maxn];
int ty[maxn];
bool divX[maxn];

void build_tree(int left, int right, pii *points) {
    if (left >= right)
        return;
    int mid = (left + right) >> 1;

    // sort(points + left, points + right + 1, divX ? cmpX :
        cmpY);
    int minx = numeric_limits<int>::max();
    int maxx = numeric_limits<int>::min();
    int miny = numeric_limits<int>::max();
    int maxy = numeric_limits<int>::min();
    for (int i = left; i < right; i++) {
        minx = min(minx, points[i].first);
        maxx = max(maxx, points[i].first);
        miny = min(miny, points[i].second);
        maxy = max(maxy, points[i].second);
    }
    divX[mid] = (maxx - minx) >= (maxy - miny);
```

```
    bool (*cmpX)(pii, pii) = [](pii a, pii b) { return a.first
        < b.first; };
    bool (*cmpY)(pii, pii) = [](pii a, pii b) { return a.second
         < b.second; };
    nth_element(points + left, points + mid, points + right,
        divX[mid] ? cmpX : cmpY);

    tx[mid] = points[mid].first;
    ty[mid] = points[mid].second;

    if (left + 1 == right)
        return;
    build_tree(left, mid, points);
    build_tree(mid + 1, right, points);
}

long long closest_dist;
int closest_node;

void find_nearest_neighbour(int left, int right, int x, int y)
    {
    if (left >= right)
        return;
    int mid = (left + right) >> 1;
    int dx = x - tx[mid];
    int dy = y - ty[mid];
    long long d = dx * (long long)dx + dy * (long long)dy;
    if (closest_dist > d && d) {
        closest_dist = d;
        closest_node = mid;
    }
    if (left + 1 == right)
        return;

    int delta = divX[mid] ? dx : dy;
    long long delta2 = delta * (long long)delta;
    int l1 = left;
    int r1 = mid;
    int l2 = mid + 1;
    int r2 = right;
    if (delta > 0)
        swap(l1, l2), swap(r1, r2);

    find_nearest_neighbour(l1, r1, x, y);
    if (delta2 < closest_dist)
        find_nearest_neighbour(l2, r2, x, y);
}

int find_nearest_neighbour(int n, int x, int y) {
    closest_dist = LLONG_MAX;
    find_nearest_neighbour(0, n, x, y);
    return closest_node;
}

// usage example
int main() {
    vector<pii> p;
    p.emplace_back(0, 2);
    p.emplace_back(0, 3);
    p.emplace_back(-1, 0);

    p.resize(unique(p.begin(), p.end()) - p.begin());

    int n = p.size();
    build_tree(0, n - 1, &p[0]);
    int res = find_nearest_neighbour(n, 0, 0);

    cout << p[res].first << " " << p[res].second << endl;

    return 0;
}
```

LinkCutTree.cpp

Description: desc

583380, 191 lines

```
// LinkCut tree with path queries. Query complexity is O(log(n)
    ) amortized.
// Based on Daniel Sleator's implementation http://www.
    codeforces.com/contest/117/submission/860934
struct Node {
    long long node_value;
    long long sub_tree_sum;
    long long add;
    bool revert;

    int size;
    Node *left;
    Node *right;
    Node *parent;

    Node(long long value)
        : node_value(value),
          sub_tree_sum(value),
          add(0),
          revert(false),
          size(1),
          left(nullptr),
          right(nullptr),
          parent(nullptr) {}

    // tests whether x is a root of a splay tree
    bool isRoot() { return parent == nullptr || (parent->left
        != this && parent->right != this); }

    void apply(long long v) {
        node_value += v;
        sub_tree_sum += v * size;
        add += v;
    }

    void push() {
        if (revert) {
            revert = false;
            Node *t = left;
            left = right;
            right = t;
            if (left != nullptr)
                left->revert = !left->revert;
            if (right != nullptr)
                right->revert = !right->revert;
        }
        if (add != 0) {
            if (left != nullptr)
                left->apply(add);
            if (right != nullptr)
                right->apply(add);
            add = 0;
        }
    }

    void pull() {
        sub_tree_sum = node_value + get_sub_tree_sum(left) +
            get_sub_tree_sum(right);
        size = 1 + get_size(left) + get_size(right);
    }

    static long long get_sub_tree_sum(Node *root) { return root
        == nullptr ? 0 : root->sub_tree_sum; }
    static int get_size(Node *root) { return root == nullptr ?
        0 : root->size; }
};

void connect(Node *ch, Node *p, int is_left_child) {
    if (ch != nullptr)
        ch->parent = p;
    if (is_left_child != 2) {
        if (is_left_child)
            p->left = ch;
        else
            p->right = ch;
    }
}

// rotates edge (x, x.parent)
//        g            g
//       /            /
//      p            x
//     / \    ->    / \
//    x  p.r       x.l  p
//   / \                / \
// x.l x.r           x.r p.r
void rotate(Node *x) {
    Node *p = x->parent;
    Node *g = p->parent;
    bool isRootP = p->isRoot();
    bool left_child_x = (x == p->left);

    // create 3 edges: (x.r(l),p), (p,x), (x,g)
    connect(left_child_x ? x->right : x->left, p, left_child_x)
        ;
    connect(p, x, !left_child_x);
    connect(x, g, isRootP ? 2 : (p == g->left ? 1 : 0));
    p->pull();
}

// brings x to the root, balancing tree
//
// zig-zig case
//        g                                  x
//       / \                p              / \
//      p  g.r rot(p)     /   \     rot(x) x.l  p
//     / \      ->      x       g     ->       / \
//    x  p.r           / \     / \            x.r  g
//   / \             x.l x.r p.r g.r              / \
// x.l x.r                                      p.r g.r
//
// zig-zag case
//      g                g                      x
//     / \              / \                    /   \
//    p  g.r rot(x)    x  g.r rot(x)     p          g
//   / \      ->      / \      ->      / \        / \
// p.l  x           p  x.r           p.l x.r    x.l g.r
//     / \         / \                / \      / \
//   x.l x.r     p.l x.l            p.l x.l  x.r g.r
void splay(Node *x) {
    while (!x->isRoot()) {
        Node *p = x->parent;
        Node *g = p->parent;
        if (!p->isRoot())
            g->push();
        p->push();
        x->push();
        if (!p->isRoot())
            rotate((x == p->left) == (p == g->left) ? p /*zig-
                zig*/ : x /*zig-zag*/);
        rotate(x);
```

```
    }
    x->push();
    x->pull();
}

// makes node x the root of the virtual tree, and also x
     becomes the leftmost node in its splay tree
Node *expose(Node *x) {
    Node *last = nullptr;
    for (Node *y = x; y != nullptr; y = y->parent) {
        splay(y);
        y->left = last;
        last = y;
    }
    splay(x);
    return last;
}

void make_root(Node *x) {
    expose(x);
    x->revert = !x->revert;
}

bool connected(Node *x, Node *y) {
    if (x == y)
        return true;
    expose(x);
    // now x.parent is null
    expose(y);
    return x->parent != nullptr;
}

void link(Node *x, Node *y) {
    assert(!connected(x, y));
    make_root(x);
    x->parent = y;
}

void cut(Node *x, Node *y) {
    make_root(x);
    expose(y);
    // check that exposed path consists of a single edge (y,x)
    assert(y->right == x && x->left == nullptr);
    y->right->parent = nullptr;
    y->right = nullptr;
}

long long query(Node *from, Node *to) {
    make_root(from);
    expose(to);
    return Node::get_sub_tree_sum(to);
}

void modify(Node *from, Node *to, long long delta) {
    make_root(from);
    expose(to);
    to->apply(delta);
}

// usage example
int main() {
    Node *n1 = new Node(1);
    Node *n2 = new Node(2);
    link(n1, n2);
    long long q = query(n1, n2);
    cout << q << endl;
    cut(n1, n2);
}
```

MosWithUpdates.cpp

Description: desc

a50c59, 109 lines

```
struct query {
    int l, r, index, t;
};

struct update_query {
    int pos, value, prev;
};

struct data_structure {
    int m[26];
    int cnt;

    data_structure() : m{}, cnt{0} {}

    void add(int x) {
        if (m[x]++ == 0)
            ++cnt;
    }

    void remove(int x) {
        if (--m[x] == 0)
            --cnt;
    }

    int get() { return cnt; }
};

void apply(data_structure &ds, vector<int> &a, int l, int r,
     int i, int x) {  // Change s[i] to x
    if (l <= i && i <= r) {
        ds.remove(a[i]);
        a[i] = x;
        ds.add(a[i]);
    } else {
        a[i] = x;
    }
}

void solve(istream &in, ostream &out) {
    string s;
    in >> s;
    int n = s.size();
    vector<int> a(n);
    for (int i = 0; i < n; ++i)
        a[i] = s[i] - 'a';
    vector<int> prev = a;

    vector<query> queries;
    vector<update_query> update_queries;

    int m;
    in >> m;

    for (int i = 0; i < m; ++i) {
        int type;
        in >> type;
        if (type == 1) {
            int pos;
            char c;
            in >> pos >> c;
            --pos;
            c -= 'a';
            update_queries.emplace_back(update_query{pos, c,
                 prev[pos]});
            prev[pos] = c;
        } else {
            int l, r;
            in >> l >> r;
            queries.emplace_back(query{l - 1, r - 1, (int)
                 queries.size(), (int)update_queries.size() - 1
                 });
        }
    }

    int block = (int)pow(n, 2 / 3.);
    sort(queries.begin(), queries.end(), [block](auto &q1, auto
          &q2) {
        if (q1.l / block != q2.l / block)
            return q1.l < q2.l;
        if (q1.r / block != q2.r / block)
            return q1.r < q2.r;
        return q1.t < q2.t;
    });

    int l = 0;
    int r = -1;
    int t = -1;
    vector<int> ans(queries.size());
    data_structure ds;

    for (auto &q : queries) {
        while (t < q.t)
            ++t, apply(ds, a, l, r, update_queries[t].pos,
                 update_queries[t].value);
        while (t > q.t)
            apply(ds, a, l, r, update_queries[t].pos,
                 update_queries[t].prev), --t;

        while (r < q.r)
            ds.add(a[++r]);
        while (l > q.l)
            ds.add(a[--l]);
        while (r > q.r)
            ds.remove(a[r--]);
        while (l < q.l)
            ds.remove(a[l++]);

        ans[q.index] = ds.get();
    }

    for (size_t i = 0; i < queries.size(); i++)
        out << ans[i] << endl;
}

// usage example
int main() {}
```

OrderedSet.cpp

Description: desc

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> e46bd8, 68 lines

```
#if defined(__GNUC__) && !defined(__clang__)

using namespace __gnu_pbds;

using ordered_set = tree<int, null_type, less<>, rb_tree_tag,
     tree_order_statistics_node_update>;
using ordered_map = tree<int, int, less<>, rb_tree_tag,
     tree_order_statistics_node_update>;

struct tree2d {
    vector<ordered_set> sets;

    tree2d(int n) : sets(n) {}

    void add(int x, int v) {
```

```
        for (; x < sets.size(); x |= x + 1)
            sets[x].insert(v);
    }

    void remove(int x, int v) {
        while (x < sets.size()) {
            sets[x].erase(v);
            x += x & -x;
        }
    }

    int query(int x, int y1, int y2) {
        int res = 0;
        for (; x >= 0; x = (x & (x + 1)) - 1)
            res += sets[x].order_of_key(y2 + 1) - sets[x].
                order_of_key(y1);
        return res;
    }
};

// usage example
int main() {
    ordered_set set;
    ordered_map map;

    map.insert(make_pair(1, 2));

    set.insert(1);
    set.insert(2);
    set.insert(4);
    set.insert(8);
    set.insert(16);

    cout << *set.find_by_order(1) << endl;                 // 2
    cout << *set.find_by_order(2) << endl;                 // 4
    cout << *set.find_by_order(4) << endl;                 // 16
    cout << (set.end() == set.find_by_order(6)) << endl;   //
        true
    cout << set.order_of_key(-5) << endl;                  // 0
    cout << set.order_of_key(1) << endl;                   // 0
    cout << set.order_of_key(3) << endl;                   // 2
    cout << set.order_of_key(4) << endl;                   // 2
    cout << set.order_of_key(400) << endl;                 // 5
    cout << endl;

    tree2d t(10);
    t.add(1, 5);
    t.add(3, 6);
    cout << t.query(3, 5, 6) << endl;
}

#else
int main() {}
#endif
```

QueueMin.cpp

Description: desc

a42544, 44 lines

```
// https://cp-algorithms.com/data_structures/
    stack_queue_modification.html

template <class T>
struct queue_min {
    stack<pair<T, T>> s1;
    stack<pair<T, T>> s2;

    T min() {
        return std::min(s1.empty() ? numeric_limits<T>::max() :
            s1.top().second,
                        s2.empty() ? numeric_limits<T>::max() :
                            s2.top().second);
    }

    void add_last(T x) {
        T min_value = s1.empty() ? x : std::min(x, s1.top().
            second);
        s1.emplace(x, min_value);
    }

    T remove_first() {
        if (s2.empty()) {
            while (!s1.empty()) {
                T x = s1.top().first;
                s1.pop();
                T min_value = s2.empty() ? x : std::min(x, s2.
                    top().second);
                s2.emplace(x, min_value);
            }
        }
        T x = s2.top().first;
        s2.pop();
        return x;
    }
};

// usage example
int main() {
    queue_min<int> q;
    q.add_last(2);
    q.add_last(3);
    cout << boolalpha;
    cout << (2 == q.min()) << endl;
    q.remove_first();
    cout << (3 == q.min()) << endl;
    q.add_last(1);
    cout << (1 == q.min()) << endl;
}
```

SegmentTree.cpp

Description: desc

"segment_tree.h" 3cfd60, 11 lines

```
// usage example
int main() {
    segtree t(10);
    t.modify(2, 3, 1);
    t.modify(3, 4, 2);
    cout << t.get(2, 3).mx << endl;

    vector<long long> a{1, 2, 10, 20};
    segtree tt(a);
    cout << sum_lower_bound(tt, 0, tt.n - 1, 12) << endl;
}
```

SegmentTree.h

Description: desc

12a024, 172 lines

```
struct segtree {
    struct node {
        // initial values for leaves
        long long mx = 0;
        long long sum = 0;
        long long add = 0;

        // set initial value for a leave
        void initialize(long long v) { mx = v; }

        // apply aggregate operation to the node
        void apply(int l, int r, long long v) {
            mx += v;
            sum += (r - l + 1) * v;
            add += v;
        }
    };

    // construct a node from its children
    static node unite(const node &a, const node &b) {
        node res;
        res.mx = max(a.mx, b.mx);
        res.sum = a.sum + b.sum;
        return res;
    }

    void push(int x, int l, int r) {
        int m = (l + r) >> 1;
        int y = x + ((m - l + 1) << 1);
        if (tree[x].add != 0) {
            tree[x + 1].apply(l, m, tree[x].add);
            tree[y].apply(m + 1, r, tree[x].add);
            tree[x].add = 0;
        }
    }

    void pull(int x, int y) { tree[x] = unite(tree[x + 1], tree
        [y]); }

    int n;
    vector<node> tree;

    void build(int x, int l, int r) {
        if (l == r) {
            return;
        }
        int m = (l + r) >> 1;
        int y = x + ((m - l + 1) << 1);
        build(x + 1, l, m);
        build(y, m + 1, r);
        pull(x, y);
    }

    template <class T>
    void build(int x, int l, int r, const vector<T> &v) {
        if (l == r) {
            tree[x].initialize(v[l]);
            return;
        }
        int m = (l + r) >> 1;
        int y = x + ((m - l + 1) << 1);
        build(x + 1, l, m, v);
        build(y, m + 1, r, v);
        pull(x, y);
    }

    node get(int x, int l, int r, int ll, int rr) {
        if (ll <= l && r <= rr) {
            return tree[x];
        }
        int m = (l + r) >> 1;
        int y = x + ((m - l + 1) << 1);
        push(x, l, r);
        node res;
        if (rr <= m) {
            res = get(x + 1, l, m, ll, rr);
        } else {
            if (ll > m) {
                res = get(y, m + 1, r, ll, rr);
            } else {
```

```
            res = unite(get(x + 1, l, m, ll, rr), get(y, m
                + 1, r, ll, rr));
        }
    }
    pull(x, y);
    return res;
}

template <class T>
void modify(int x, int l, int r, int ll, int rr, const T &v
    ) {
    if (ll <= l && r <= rr) {
        tree[x].apply(l, r, v);
        return;
    }
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    if (ll <= m) {
        modify(x + 1, l, m, ll, rr, v);
    }
    if (rr > m) {
        modify(y, m + 1, r, ll, rr, v);
    }
    pull(x, y);
}

segtree(int _n) : n(_n) {
    assert(n > 0);
    tree.resize(2 * n - 1);
    build(0, 0, n - 1);
}

template <class T>
segtree(const vector<T> &v) {
    n = v.size();
    assert(n > 0);
    tree.resize(2 * n - 1);
    build(0, 0, n - 1, v);
}

node get(int ll, int rr) {
    assert(0 <= ll && ll <= rr && rr <= n - 1);
    return get(0, 0, n - 1, ll, rr);
}

node get(int p) {
    assert(0 <= p && p <= n - 1);
    return get(0, 0, n - 1, p, p);
}

template <class T>
void modify(int ll, int rr, const T v) {
    assert(0 <= ll && ll <= rr && rr <= n - 1);
    modify(0, 0, n - 1, ll, rr, v);
}

int find_first(int ll, int rr, const function<bool(const
    node &)> &f, int x, int l, int r) {
    if (ll <= l && r <= rr && !f(tree[x])) {
        return -1;
    }
    if (l == r) {
        return l;
    }
    push(x, l, r);
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    int res = -1;
    if (ll <= m) {
        res = find_first(ll, rr, f, x + 1, l, m);
    }
    if (rr > m && res == -1) {
        res = find_first(ll, rr, f, y, m + 1, r);
    }
    pull(x, y);
    return res;
}

// calls all FALSE elements to the left of the sought
    position exactly once
int find_first(int ll, int rr, const function<bool(const
    node &)> &f) {
    assert(0 <= ll && ll <= rr && rr <= n - 1);
    return find_first(ll, rr, f, 0, 0, n - 1);
}
};

// Returns min(p | p<=rr && sum[ll..p]>=sum). If no such p
    exists, returns -1
int sum_lower_bound(segtree &t, int ll, int rr, long long sum)
    {
    long long sumSoFar = 0;
    return t.find_first(ll, rr, [&](const segtree::node &node)
        {
        if (sumSoFar + node.sum >= sum)
            return true;
        sumSoFar += node.sum;
        return false;
    });
}
```

SegmentTreeWithoutRecursion.cpp

Description: desc

841166, 30 lines

```
int get(const vector<int> &t, int i) {
    return t[i + t.size() / 2];
}

void add(vector<int> &t, int i, int value) {
    i += t.size() / 2;
    t[i] += value;
    for (; i > 1; i >>= 1)
        t[i >> 1] = min(t[i], t[i ^ 1]);
}

int min(const vector<int> &t, int a, int b) {
    int res = numeric_limits<int>::max();
    for (a += t.size() / 2, b += t.size() / 2; a <= b; a = (a +
        1) >> 1, b = (b - 1) >> 1) {
        if ((a & 1) != 0)
            res = min(res, t[a]);
        if ((b & 1) == 0)
            res = min(res, t[b]);
    }
    return res;
}

// usage example
int main() {
    int n = 10;
    vector<int> t(n + n);
    add(t, 0, -1);
    add(t, 9, -2);
    cout << (-2 == min(t, 0, 9)) << endl;
}
```

Sparse-segment-tree.cpp

Description: desc

d3487d, 50 lines

```
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int left;
    int right;
    int nsum;

    Node(int lo, int hi, int val) : left(lo), right(hi), nsum(
        val) {}

    void add(int pos, int val) {
        if (pos < left || pos > right) {
            return;
        }
        nsum += val;
        if (right - left == 1) {
            return;
        }
        int mid = (left + right) / 2;
        if (pos < mid) {
            if (l == nullptr) {
                l = new Node(left, mid, 0);
            }
            l->add(pos, val);
        } else {
            if (r == nullptr) {
                r = new Node(mid + 1, right, 0);
            }
            r->add(pos, val);
        }
    }

    int sum(int from, int to) {
        if (to < left || right < from) {
            return 0;
        }
        if (from <= left && right <= to) {
            return nsum;
        }
        return (l ? l->sum(from, to) : 0) + (r ? r->sum(from,
            to) : 0);
    }
};

// usage example
int main() {
    Node t(0, 1000, 0);
    t.add(1, 1);
    t.add(100, 2);
    cout << t.sum(0, 10) << endl;
    cout << t.sum(0, 200) << endl;
}
```

Sparse-table.cpp

Description: desc

79d271, 37 lines

```
#ifdef _MSC_VER
int __builtin_clz(unsigned x) {
    int bit = 31;
    while (bit >= 0 && (x & (1 << bit)) == 0)
        --bit;
    return 31 - bit;
}
#endif

template <class T, class F = function<T(const T &, const T &)>>
struct SparseTable {
```

```
    vector<vector<T>> t;
    F func;

    SparseTable(const vector<T> &a, F f) : t(32 - __builtin_clz
         (a.size())), func(std::move(f)) {
        t[0] = a;
        for (size_t i = 1; i < t.size(); i++) {
            t[i].resize(a.size() - (1 << i) + 1);
            for (size_t j = 0; j < t[i].size(); j++)
                t[i][j] = func(t[i - 1][j], t[i - 1][j + (1 <<
                     (i - 1))]);
        }
    }

    T get(int from, int to) const {
        assert(0 <= from && from <= to && to <= (int)t[0].size
             () - 1);
        int k = 31 - __builtin_clz(to - from + 1);
        return func(t[k][from], t[k][to - (1 << k) + 1]);
    }
};

// usage example
int main() {
    vector<int> a{3, 2, 1};
    SparseTable<int> st(a, [](int i, int j) { return min(i, j);
         });
    cout << st.get(0, 2) << endl;
    cout << st.get(0, 1) << endl;
}
```

Treap.cpp

Description: desc

"treap.h" 5b0a7a, 18 lines

```
// usage example
int main() {
    pTreap t = nullptr;
    int pos = 0;
    for (int a : {1, 2, 7, 4, 5})
        insert(t, pos++, a);
    int n = t->size;
    for (int i = 0; i < n; ++i)
        cout << query(t, i, i).node_value << endl;
    modify(t, 1, 3, 10);
    for (int i = 0; i < n; ++i)
        cout << query(t, i, i).node_value << endl;
    for (int i = 0; i < n; ++i)
        remove(t, 0);
    cout << Treap::get_size(t) << endl;

    clear(t);
}
```

Treap.h

Description: desc

5645e3, 137 lines

```
// https://cp-algorithms.com/data_structures/treap.html

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
    count());

struct Treap {
    long long node_value;
    long long mx;
    long long sum;
    long long add;

    long long key;  // keys should be unique
    int size;
    long long prio;
    Treap *l, *r;

    Treap(long long key, long long value)
        : node_value(value), mx(value), sum(value), add(0), key
            (key), size(1), prio(rng()), l(nullptr), r(nullptr
            ) {}

    void apply(long long v) {
        node_value += v;
        mx += v;
        sum += v * size;
        add += v;
    }

    void push() {
        if (add != 0) {
            if (l != nullptr)
                l->apply(add);
            if (r != nullptr)
                r->apply(add);
            add = 0;
        }
    }

    void pull() {
        mx = max(node_value, max(get_mx(l), get_mx(r)));
        sum = node_value + get_sum(l) + get_sum(r);
        size = 1 + get_size(l) + get_size(r);
    }

    static long long get_mx(Treap *root) { return root ==
         nullptr ? numeric_limits<long long>::min() : root->mx;
          }

    static long long get_sum(Treap *root) { return root ==
         nullptr ? 0 : root->sum; }

    static int get_size(Treap *root) { return root == nullptr ?
          0 : root->size; }
};

using pTreap = Treap *;

void split(pTreap t, long long min_right, pTreap &l, pTreap &r)
     {
    if (!t) {
        l = r = nullptr;
    } else {
        t->push();
        if (t->key >= min_right) {
            split(t->l, min_right, l, t->l);
            r = t;
        } else {
            split(t->r, min_right, t->r, r);
            l = t;
        }
        t->pull();
    }
}

void merge(pTreap &t, pTreap &l, pTreap &r) {
    if (!l || !r) {
        t = l ? l : r;
    } else {
        l->push();
        r->push();
        if (l->prio > r->prio) {
            merge(l->r, l->r, r);
            t = l;
        } else {
            merge(r->l, l, r->l);
            t = r;
        }
        t->pull();
    }
}

void insert(pTreap &t, long long key, long long value) {
    pTreap l, r;
    split(t, key, l, r);
    auto node = new Treap(key, value);
    merge(t, l, node);
    merge(t, t, r);
}

void remove(pTreap &t, long long key) {
    pTreap left1, right1, left2, right2;
    split(t, key, left1, right1);
    split(right1, key + 1, left2, right2);
    delete left2;
    merge(t, left1, right2);
}

void modify(pTreap &t, long long ll, long long rr, long long
     delta) {
    pTreap left1, right1, left2, right2;
    split(t, rr + 1, left1, right1);
    split(left1, ll, left2, right2);
    if (right2 != nullptr)
        right2->apply(delta);
    merge(t, left2, right2);
    merge(t, t, right1);
}

Treap query(pTreap &t, long long ll, long long rr) {
    pTreap left1, right1, left2, right2;
    split(t, rr + 1, left1, right1);
    split(left1, ll, left2, right2);
    Treap res(0, 0);
    if (right2)
        res = *right2;
    merge(t, left2, right2);
    merge(t, t, right1);
    return res;
}

void clear(pTreap &t) {
    if (!t)
        return;
    clear(t->l);
    clear(t->r);
    delete t;
    t = nullptr;
}

void print(pTreap t) {
    if (!t)
        return;
    print(t->l);
    cout << t->node_value << endl;
    print(t->r);
}
```

TreapBst.cpp

Description: desc

a99a35, 115 lines

```
// https://cp-algorithms.com/data_structures/treap.html
```

```
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
    count());

struct Treap {
    int key;
    long long prio;
    int size;
    Treap *l, *r;

    Treap(int key) : key(key), prio(rng()), size(1), l(nullptr)
        , r(nullptr) {}

    void update() { size = 1 + get_size(l) + get_size(r); }

    static int get_size(Treap *node) { return node ? node->size
        : 0; }
};

using pTreap = Treap *;

void split(pTreap t, int key, pTreap &l, pTreap &r) {
    if (!t)
        l = r = nullptr;
    else if (key < t->key)
        split(t->l, key, l, t->l), r = t, t->update();
    else
        split(t->r, key, t->r, r), l = t, t->update();
}

void merge(pTreap &t, pTreap l, pTreap r) {
    if (!l || !r)
        t = l ? l : r;
    else if (l->prio > r->prio)
        merge(l->r, l->r, r), t = l, t->update();
    else
        merge(r->l, l, r->l), t = r, t->update();
}

void insert(pTreap &t, pTreap it) {
    if (!t)
        t = it;
    else if (it->prio > t->prio)
        split(t, it->key, it->l, it->r), t = it, t->update();
    else
        insert(it->key < t->key ? t->l : t->r, it), t->update()
            ;
}

void erase(pTreap &t, int key) {
    if (t->key == key) {
        pTreap l = t->l;
        pTreap r = t->r;
        delete t;
        merge(t, l, r);
    } else {
        erase(key < t->key ? t->l : t->r, key), t->update();
    }
}

pTreap unite(pTreap l, pTreap r) {
    if (!l || !r)
        return l ? l : r;
    if (l->prio < r->prio)
        swap(l, r);
    pTreap lt, rt;
    split(r, l->key, lt, rt);
    l->l = unite(l->l, lt);
    l->r = unite(l->r, rt);
    return l;
}

int kth(pTreap t, int k) {
    if (k < Treap::get_size(t->l))
        return kth(t->l, k);
    else if (k > Treap::get_size(t->l))
        return kth(t->r, k - Treap::get_size(t->l) - 1);
    return t->key;
}

void print(pTreap t) {
    if (!t)
        return;
    print(t->l);
    cout << t->key << endl;
    print(t->r);
}

void clear(pTreap &t) {
    if (!t)
        return;
    clear(t->l);
    clear(t->r);
    delete t;
    t = nullptr;
}

// usage example
int main() {
    pTreap t1 = nullptr;
    int a1[] = {1, 2};
    for (int x : a1)
        insert(t1, new Treap(x));

    pTreap t2 = nullptr;
    int a2[] = {7, 4, 5};
    for (int x : a2)
        insert(t2, new Treap(x));

    pTreap t = nullptr;
    merge(t, t1, t2);

    for (int i = 0; i < t->size; ++i) {
        cout << kth(t, i) << endl;
    }

    clear(t);
}
```

TreapBstSharedPtr.cpp

Description: desc

cafc36, 92 lines

```
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
    count());

struct Treap;

using pTreap = shared_ptr<Treap>;

struct Treap {
    int key;
    long long prio;
    int size;
    pTreap l, r;

    Treap(int key) : key(key), prio(rng()), size(1), l(nullptr)
        , r(nullptr) {}

    void update() { size = 1 + get_size(l) + get_size(r); }

    static int get_size(pTreap node) { return node ? node->size
        : 0; }

    //    ~item() {
    //        cout << "item " << key << " disposed" << endl;
    //    }
};

void split(pTreap t, int key, pTreap &l, pTreap &r) {
    if (!t)
        l = r = nullptr;
    else if (key < t->key)
        split(t->l, key, l, t->l), r = t, t->update();
    else
        split(t->r, key, t->r, r), l = t, t->update();
}

void merge(pTreap &t, pTreap &l, pTreap &r) {
    if (!l || !r)
        t = l ? l : r;
    else if (l->prio > r->prio)
        merge(l->r, l->r, r), t = l, t->update();
    else
        merge(r->l, l, r->l), t = r, t->update();
}

void insert(pTreap &t, pTreap it) {
    if (!t)
        t = it;
    else if (it->prio > t->prio)
        split(t, it->key, it->l, it->r), t = it, t->update();
    else
        insert(it->key < t->key ? t->l : t->r, it), t->update()
            ;
}

void erase(pTreap &t, int key) {
    if (t->key == key)
        merge(t, t->l, t->r);
    else
        erase(key < t->key ? t->l : t->r, key), t->update();
}

int kth(pTreap &t, int k) {
    if (k < Treap::get_size(t->l))
        return kth(t->l, k);
    else if (k > Treap::get_size(t->l))
        return kth(t->r, k - Treap::get_size(t->l) - 1);
    return t->key;
}

void print(pTreap &t) {
    if (!t)
        return;
    print(t->l);
    cout << t->key << endl;
    print(t->r);
}

// usage example
int main() {
    pTreap t1 = nullptr;
    int a1[] = {1, 2};
    for (int x : a1)
        insert(t1, make_shared<Treap>(x));

    pTreap t2 = nullptr;
```

```
    int a2[] = {7, 4, 5};
    for (int x : a2)
        insert(t2, make_shared<Treap>(x));

    pTreap t = nullptr;
    merge(t, t1, t2);

    for (int i = 0; i < t->size; ++i) {
        cout << kth(t, i) << endl;
    }
}
```

TreapIndexed.cpp

Description: desc

0039b2, 201 lines

```
// https://cp-algorithms.com/data_structures/treap.html

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
    count());

struct Treap {
    long long node_value;
    long long mx;
    long long sum;
    long long add;

    int size;
    long long prio;
    Treap *l, *r;

    Treap(long long value)
        : node_value(value), mx(value), sum(value), add(0),
          size(1), prio(rng()), l(nullptr), r(nullptr) {}

    void apply(long long v) {
        node_value += v;
        mx += v;
        sum += v * size;
        add += v;
    }

    void push() {
        if (add != 0) {
            if (l != nullptr)
                l->apply(add);
            if (r != nullptr)
                r->apply(add);
            add = 0;
        }
    }

    void pull() {
        mx = max(node_value, max(get_mx(l), get_mx(r)));
        sum = node_value + get_sum(l) + get_sum(r);
        size = 1 + get_size(l) + get_size(r);
    }

    static long long get_mx(Treap *root) { return root ==
        nullptr ? numeric_limits<long long>::min() : root->mx;
        }

    static long long get_sum(Treap *root) { return root ==
        nullptr ? 0 : root->sum; }

    static int get_size(Treap *root) { return root == nullptr ?
        0 : root->size; }
};

using pTreap = Treap *;

void split(pTreap t, int min_right, pTreap &l, pTreap &r) {
    if (!t) {
        l = r = nullptr;
    } else {
        t->push();
        if (Treap::get_size(t->l) >= min_right) {
            split(t->l, min_right, l, t->l);
            r = t;
        } else {
            split(t->r, min_right - Treap::get_size(t->l) - 1,
                t->r, r);
            l = t;
        }
        t->pull();
    }
}

void merge(pTreap &t, pTreap &l, pTreap &r) {
    if (!l || !r) {
        t = l ? l : r;
    } else {
        l->push();
        r->push();
        if (l->prio > r->prio) {
            merge(l->r, l->r, r);
            t = l;
        } else {
            merge(r->l, l, r->l);
            t = r;
        }
        t->pull();
    }
}

void insert(pTreap &t, int index, long long value) {
    pTreap l, r;
    split(t, index, l, r);
    auto node = new Treap(value);
    merge(t, l, node);
    merge(t, t, r);
}

void remove(pTreap &t, int index) {
    pTreap left1, right1, left2, right2;
    split(t, index, left1, right1);
    split(right1, 1, left2, right2);
    delete left2;
    merge(t, left1, right2);
}

void modify(pTreap &t, int ll, int rr, long long delta) {
    pTreap left1, right1, left2, right2;
    split(t, rr + 1, left1, right1);
    split(left1, ll, left2, right2);
    if (right2 != nullptr)
        right2->apply(delta);
    merge(t, left2, right2);
    merge(t, t, right1);
}

Treap query(pTreap &t, int ll, int rr) {
    pTreap left1, right1, left2, right2;
    split(t, rr + 1, left1, right1);
    split(left1, ll, left2, right2);
    Treap res = *right2;
    merge(t, left2, right2);
    merge(t, t, right1);
    return res;
}

int find_first(pTreap t, int ll, int rr, const function<bool(
    const Treap &)> &f, int l, int r) {
    if (ll <= l && r <= rr && !f(*t)) {
        return -1;
    }
    if (l == r) {
        return l;
    }
    t->push();
    int m = Treap::get_size(t->l);
    int res = -1;
    if (ll < m) {
        res = find_first(t->l, ll, rr, f, l, l + m - 1);
    }
    if (res == -1) {
        auto single = new Treap(0);
        single->size = 1;
        single->apply(t->node_value);
        res = find_first(single, ll, rr, f, l + m, l + m);
    }
    if (rr > m && res == -1) {
        res = find_first(t->r, ll, rr, f, l + m + 1, r);
    }
    t->pull();
    return res;
}

// calls all FALSE elements to the left of the sought position
    exactly once
int find_first(pTreap t, int ll, int rr, const function<bool(
    const Treap &)> &f) {
    assert(0 <= ll && ll <= rr && rr <= Treap::get_size(t) - 1)
        ;
    return find_first(t, ll, rr, f, 0, Treap::get_size(t) - 1);
}

// Returns min(p | p<=rr && sum[ll..p]>=sum). If no such p
    exists, returns -1
int sum_lower_bound(pTreap t, int ll, int rr, long long sum) {
    long long sumSoFar = 0;
    return find_first(t, ll, rr, [&](const Treap &node) {
        if (sumSoFar + node.sum >= sum)
            return true;
        sumSoFar += node.sum;
        return false;
    });
}

void clear(pTreap &t) {
    if (!t)
        return;
    clear(t->l);
    clear(t->r);
    delete t;
    t = nullptr;
}

void print(pTreap t) {
    if (!t)
        return;
    print(t->l);
    cout << t->node_value << endl;
    print(t->r);
}

// usage example
int main() {
```

```
    pTreap t = nullptr;
    int pos = 0;
    for (int a : {1, 2, 7, 4, 5})
        insert(t, pos++, a);
    int n = t->size;
    for (int i = 0; i < n; ++i)
        cout << query(t, i, i).node_value << endl;
    modify(t, 1, 3, 10);
    for (int i = 0; i < n; ++i)
        cout << query(t, i, i).node_value << endl;
    for (int i = 0; i < n; ++i)
        remove(t, 0);
    cout << Treap::get_size(t) << endl;

    for (int v : {2, 1, 10, 20}) {
        insert(t, Treap::get_size(t), v);
    }
    cout << (2 == sum_lower_bound(t, 0, Treap::get_size(t) - 1,
         12));

    clear(t);
}
```

Tree2d.cpp

Description: desc

"treap.h" 948270, 38 lines

```
struct tree_2d {
    vector<pTreap> t;

    tree_2d(int n) : t(2 * n) {}

    long long query(int x1, int x2, int y1, int y2) {
        long long res = 0;
        for (x1 += t.size() / 2, x2 += t.size() / 2; x1 <= x2;
             x1 = (x1 + 1) >> 1, x2 = (x2 - 1) >> 1) {
            if ((x1 & 1) != 0)
                res += ::query(t[x1], y1, y2).sum;
            if ((x2 & 1) == 0)
                res += ::query(t[x2], y1, y2).sum;
        }
        return res;
    }

    void insert(int x, int y, int value) {
        x += t.size() / 2;
        for (; x > 0; x >>= 1)
            ::insert(t[x], y, value);
    }

    void remove(int x, int y) {
        x += t.size() / 2;
        for (; x > 0; x >>= 1)
            ::remove(t[x], y);
    }
};

// usage example
int main() {
    tree_2d t(10);
    t.insert(1, 5, 3);
    t.insert(3, 3, 2);
    t.insert(2, 6, 1);
    t.remove(2, 6);
    cout << t.query(0, 9, 0, 9) << endl;
}
```

WaveletTree.cpp

Description: desc

538158, 82 lines

```
struct wavelet_tree {
    int lo, hi;
    wavelet_tree *l = nullptr;
    wavelet_tree *r = nullptr;
    vector<int> b;

    static wavelet_tree create(int *from, int *to) {
        int min = *min_element(from, to);
        int max = *max_element(from, to);
        return wavelet_tree(from, to, min, max);
    }

    // nos are in range [x,y]
    // array indices are [from, to)
    wavelet_tree(int *from, int *to, int x, int y) {
        lo = x;
        hi = y;
        if (lo == hi || from >= to)
            return;
        int mid = (lo + hi) / 2;
        auto f = [mid](int x) { return x <= mid; };
        b.reserve(to - from + 1);
        b.push_back(0);
        for (auto it = from; it != to; it++) {
            b.push_back(b.back() + f(*it));
        }
        // see how lambda function is used here
        auto pivot = stable_partition(from, to, f);
        l = new wavelet_tree(from, pivot, lo, mid);
        r = new wavelet_tree(pivot, to, mid + 1, hi);
    }

    // kth smallest element in [l, r]
    int kth(int l, int r, int k) {
        if (l > r)
            return 0;
        if (lo == hi)
            return lo;
        int inLeft = b[r] - b[l - 1];
        int lb = b[l - 1];  // amt of nos in first (l-1) nos
             that go in left
        int rb = b[r];      // amt of nos in first (r) nos that
              go in left
        return k <= inLeft ? this->l->kth(lb + 1, rb, k) : this
             ->r->kth(l - lb, r - rb, k - inLeft);
    }

    // count of nos in [l, r] Less than or equal to k
    int LTE(int l, int r, int k) {
        if (l > r || k < lo)
            return 0;
        if (hi <= k)
            return r - l + 1;
        int lb = b[l - 1];
        int rb = b[r];
        return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l -
             lb, r - rb, k);
    }

    // count of nos in [l, r] equal to k
    int count(int l, int r, int k) {
        if (l > r || k < lo || k > hi)
            return 0;
        if (lo == hi)
            return r - l + 1;
        int lb = b[l - 1];
        int rb = b[r];
        int mid = (lo + hi) / 2;
        return k <= mid ? this->l->count(lb + 1, rb, k) : this
             ->r->count(l - lb, r - rb, k);
    }

    ~wavelet_tree() {
        delete l;
        delete r;
    }
};

// usage example
int main() {
    int a[] = {3, 1, 4, 2};
    wavelet_tree wtree = wavelet_tree::create(a, a + 4);

    cout << wtree.kth(1, 3, 2) << endl;
    cout << wtree.LTE(1, 3, 2) << endl;
    cout << wtree.count(1, 4, 2) << endl;
}
```

backtrack (10)

Mis.cpp

Description: desc

e20033, 44 lines

```
#ifdef _MSC_VER
int __builtin_ctzll(unsigned long long x) {
    int bit = 0;
    while (bit < 64 && (x & (1LL << bit)) == 0)
        ++bit;
    return bit;
}
int __builtin_popcountll(unsigned long long x) {
    int bits = 0;
    for (; x; x &= x - 1, ++bits)
        ;
    return bits;
}
#endif

using ll = long long;

int ctz(ll x) {
    return x == 0 ? 64 : __builtin_ctzll(x);
}

// maximum independent set in O(3^(n/3))
int mis(const vector<ll> &g, ll unused) {
    if (unused == 0)
        return 0;
    int v = -1;
    for (int u = __builtin_ctzll(unused); u < g.size(); u +=
         ctz(unused >> (u + 1)) + 1)
        if (v == -1 || __builtin_popcountll(g[v] & unused) >
             __builtin_popcountll(g[u] & unused))
            v = u;
    int res = 0;
    ll nv = g[v] & unused;
    for (int y = __builtin_ctzll(nv); y < g.size(); y += ctz(nv
          >> (y + 1)) + 1)
        res = max(res, 1 + mis(g, unused & ~g[y]));
    return res;
}

// usage example
int main() {
```

```
    vector<ll> g{0b110, 0b101, 0b011};
    for (int i = 0; i < g.size(); ++i) {
        g[i] |= 1LL << i;
    }
    cout << mis(g, (1LL << g.size()) - 1) << endl;
}
```

combinatorics (11)

Binomial.cpp

Description: desc

7b4420, 11 lines

```
int binomial(int n, int k) {
    int res = 1;
    for (int i = 0; i < k; i++) {
        res = res * (n - i) / (i + 1);
    }
    return res;
}

int main() {
    cout << binomial(5, 3) << endl;
}
```

EnumeratingCombinations.cpp

Description: desc

a39dff, 22 lines

```
bool next_combination(vector<int> &comb, int n) {
    int k = comb.size();
    for (int i = k - 1; i >= 0; i--) {
        if (comb[i] < n - k + i) {
            ++comb[i];
            while (++i < k) {
                comb[i] = comb[i - 1] + 1;
            }
            return true;
        }
    }
    return false;
}

int main() {
    vector<int> comb{0, 1, 2};
    do {
        for (int v : comb)
            cout << v + 1 << " ";
        cout << endl;
    } while (next_combination(comb, 5));
}
```

geometry (12)

AngleAreaOrientationSortRotationPerpendicular.cpp

Description: desc

351302, 72 lines

```
using ll = long long;
#define PI acos(-1)

// pay attention to case ax==0 && ay==0 or bx==0 && by == 0
double angle_between(ll ax, ll ay, ll bx, ll by) {
    double a = atan2(ax * by - ay * bx, ax * bx + ay * by);
    return a < 0 ? a + 2 * PI : a;
}

// pay attention to case ax==0 && ay==0 or bx==0 && by == 0
double angle_between2(ll ax, ll ay, ll bx, ll by) {
    double a = atan2(by, bx) - atan2(ay, ax);
    return a < 0 ? a + 2 * PI : a;
}

ll double_signed_area(const vector<int> &x, const vector<int> &
    y) {
    int n = x.size();
    ll area = 0;
    for (int i = 0, j = n - 1; i < n; j = i++) {
        area += (ll)(x[i] - x[j]) * (y[i] + y[j]);  // area +=
            (long) x[i] * y[j] - (long) x[j] * y[i];
    }
    return area;
}

// Returns -1 for clockwise, 0 for straight line, 1 for
    counterclockwise orientation
int orientation(ll ax, ll ay, ll bx, ll by, ll cx, ll cy) {
    bx -= ax;
    by -= ay;
    cx -= ax;
    cy -= ay;
    ll det = bx * cy - by * cx;
    return (det > 0) - (det < 0);
}

bool is_middle(ll a, ll m, ll b) {
    return min(a, b) <= m && m <= max(a, b);
}

bool is_middle(ll ax, ll ay, ll mx, ll my, ll bx, ll by) {
    return orientation(ax, ay, mx, my, bx, by) == 0 &&
        is_middle(ax, mx, bx) && is_middle(ay, my, by);
}

struct Point {
    int x, y;

    bool operator<(const Point &o) const {
        bool up1 = y > 0 || (y == 0 && x >= 0);
        bool up2 = o.y > 0 || (o.y == 0 && o.x >= 0);
        if (up1 != up2)
            return up1;
        ll cmp = (ll)o.x * y - (ll)o.y * x;
        if (cmp != 0)
            return cmp < 0;
        return (ll)x * x + (ll)y * y < (ll)o.x * o.x + (ll)o.y
            * o.y;
        // return atan2(y, x) < atan2(o.y, o.x);
    }
};

pair<double, double> rotate_ccw(pair<double, double> p, double
    angle) {
    return {p.first * cos(angle) - p.second * sin(angle), p.
        first * sin(angle) + p.second * cos(angle)};
}

struct Line {
    ll a, b, c;
};

Line perpendicular(Line line, ll x, ll y) {
    return {-line.b, line.a, line.b * x - line.a * y};
}

// usage example
int main() {}
```

ConvexHull.cpp

Description: desc

dc0b85, 34 lines

```
// Convex hull construction in O(n*log(n)): https://cp-
    algorithms.com/geometry/grahams-scan-convex-hull.html

struct point {
    int x, y;
};

bool isNotRightTurn(const point &a, const point &b, const point
    &c) {
    long long cross = (long long)(a.x - b.x) * (c.y - b.y) - (
        long long)(a.y - b.y) * (c.x - b.x);
    long long dot = (long long)(a.x - b.x) * (c.x - b.x) + (
        long long)(a.y - b.y) * (c.y - b.y);
    return cross < 0 || (cross == 0 && dot <= 0);
}

vector<point> convex_hull(vector<point> points) {
    sort(points.begin(), points.end(), [](auto a, auto b) {
        return a.x < b.x || (a.x == b.x && a.y < b.y); });
    int n = points.size();
    vector<point> hull;
    for (int i = 0; i < 2 * n - 1; i++) {
        int j = i < n ? i : 2 * n - 2 - i;
        while (hull.size() >= 2 && isNotRightTurn(hull.end()
            [-2], hull.end()[-1], points[j]))
            hull.pop_back();
        hull.push_back(points[j]);
    }
    hull.pop_back();
    return hull;
}

// usage example
int main() {
    vector<point> hull1 = convex_hull({{0, 0}, {3, 0}, {0, 3},
        {1, 1}});
    cout << (3 == hull1.size()) << endl;

    vector<point> hull2 = convex_hull({{0, 0}, {0, 0}});
    cout << (1 == hull2.size()) << endl;
}
```

Diameter.cpp

Description: desc

6fd903, 57 lines

```
typedef pair<double, double> point;

bool cw(const point &a, const point &b, const point &c) {
    return (b.first - a.first) * (c.second - a.second) - (b.
        second - a.second) * (c.first - a.first) < 0;
}

vector<point> convexHull(vector<point> p) {
    int n = p.size();
    if (n <= 1)
        return p;
    int k = 0;
    sort(p.begin(), p.end());
    vector<point> q(n * 2);
    for (int i = 0; i < n; q[k++] = p[i++])
        for (; k >= 2 && !cw(q[k - 2], q[k - 1], p[i]); --k)
            ;
    for (int i = n - 2, t = k; i >= 0; q[k++] = p[i--])
        for (; k > t && !cw(q[k - 2], q[k - 1], p[i]); --k)
            ;
    q.resize(k - 1 - (q[0] == q[1]));
    return q;
```

```
}

double area(const point &a, const point &b, const point &c) {
    return abs((b.first - a.first) * (c.second - a.second) - (b
        .second - a.second) * (c.first - a.first));
}

double dist(const point &a, const point &b) {
    return hypot(a.first - b.first, a.second - b.second);
}

double diameter(const vector<point> &p) {
    vector<point> h = convexHull(p);
    int m = h.size();
    if (m == 1)
        return 0;
    if (m == 2)
        return dist(h[0], h[1]);
    int k = 1;
    while (area(h[m - 1], h[0], h[(k + 1) % m]) > area(h[m -
        1], h[0], h[k]))
        ++k;
    double res = 0;
    for (int i = 0, j = k; i <= k && j < m; i++) {
        res = max(res, dist(h[i], h[j]));
        while (j < m && area(h[i], h[(i + 1) % m], h[(j + 1) %
            m]) > area(h[i], h[(i + 1) % m], h[j])) {
            res = max(res, dist(h[i], h[(j + 1) % m]));
            ++j;
        }
    }
    return res;
}

// usage example
int main() {
    double d = diameter({{0, 0}, {3, 0}, {0, 3}, {1, 1}});
    cout << d << endl;
}
```

DynamicUpperEnvelope.cpp

Description: desc

e66a8a, 60 lines

```
// Container where you can add lines of the form ax+b, and
// get_max maximum values at points x. For each line, also
    keeps a value p,
// which is the last (maximum) point for which the current line
    is dominant.
// (obviously, for the last line, p is infinity) Useful for
    dynamic programming.

using ll = long long;

struct Line {
    ll a, b;
    mutable ll p;

    bool operator<(const Line &o) const { return a < o.a; }

    bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    const ll inf = numeric_limits<ll>::max();

    ll div(ll a, ll b) {  // floored division
        return a / b - ((a ^ b) < 0 && a % b);
    }

    bool isect(iterator x, iterator y) {
        if (y == end()) {
            x->p = inf;
            return false;
        }
        if (x->a == y->a)
            x->p = x->b > y->b ? inf : -inf;
        else
            x->p = div(y->b - x->b, x->a - y->a);
        return x->p >= y->p;
    }

    void add_line(ll a, ll b) {
        auto z = insert({a, b, 0}), y = z++, x = y;
        while (isect(y, z))
            z = erase(z);
        if (x != begin() && isect(--x, y))
            isect(x, erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }

    ll get_max(ll x) {
        assert(!empty());
        auto line = *lower_bound(x);
        return line.a * x + line.b;
    }
};

// usage example
int main() {
    LineContainer lc;
    lc.add_line(1, 3);
    lc.add_line(2, 1);
    cout << lc.get_max(1) << endl;
}
```

FindSegmentsIntersection.cpp

Description: desc

<utility> d1b7c0, 98 lines

```
typedef pair<int, int> pii;

int cross(int ax, int ay, int bx, int by, int cx, int cy) {
    return (bx - ax) * (cy - ay) - (by - ay) * (cx - ax);
}

int cross(pii a, pii b, pii c) {
    return cross(a.first, a.second, b.first, b.second, c.first,
        c.second);
}

struct segment {
    pii a, b;
    int id;

    segment(pii a, pii b, int id) : a(std::move(a)), b(std::
        move(b)), id(id) {}

    bool operator<(const segment &o) const {
        if (a.first < o.a.first) {
            int s = cross(a, b, o.a);
            return (s > 0 || (s == 0 && a.second < o.a.second))
                ;
        } else if (a.first > o.a.first) {
            int s = cross(o.a, o.b, a);
            return (s < 0 || (s == 0 && a.second < o.a.second))
                ;
        }
        return a.second < o.a.second;
    }
};

struct event {
    pii p;
    int id;
    int type;

    event(pii p, int id, int type) : p(std::move(p)), id(id),
        type(type) {}

    bool operator<(const event &o) const {
        return p.first < o.p.first ||
               (p.first == o.p.first && (type > o.type || (type
                   == o.type && p.second < o.p.second)));
    }
};

bool intersect(segment s1, segment s2) {
    int x1 = s1.a.first, y1 = s1.a.second, x2 = s1.b.first, y2
        = s1.b.second;
    int x3 = s2.a.first, y3 = s2.a.second, x4 = s2.b.first, y4
        = s2.b.second;
    if (max(x1, x2) < min(x3, x4) || max(x3, x4) < min(x1, x2)
        || max(y1, y2) < min(y3, y4) ||
        max(y3, y4) < min(y1, y2)) {
        return false;
    }
    int z1 = (x3 - x1) * (y2 - y1) - (y3 - y1) * (x2 - x1);
    int z2 = (x4 - x1) * (y2 - y1) - (y4 - y1) * (x2 - x1);
    if ((z1 < 0 && z2 < 0) || (z1 > 0 && z2 > 0)) {
        return false;
    }
    int z3 = (x1 - x3) * (y4 - y3) - (y1 - y3) * (x4 - x3);
    int z4 = (x2 - x3) * (y4 - y3) - (y2 - y3) * (x4 - x3);
    if ((z3 < 0 && z4 < 0) || (z3 > 0 && z4 > 0)) {
        return false;
    }
    return true;
}

pii findIntersection(vector<segment> s) {
    int n = s.size();
    vector<event> e;
    for (int i = 0; i < n; ++i) {
        if (s[i].a > s[i].b)
            swap(s[i].a, s[i].b);
        e.emplace_back(s[i].a, i, 1);
        e.emplace_back(s[i].b, i, -1);
    }
    sort(e.begin(), e.end());

    set<segment> q;

    for (int i = 0; i < n * 2; ++i) {
        int id = e[i].id;
        if (e[i].type == 1) {
            auto it = q.lower_bound(s[id]);
            if (it != q.end() && intersect(*it, s[id]))
                return {it->id, s[id].id};
            if (it != q.begin() && intersect(*--it, s[id]))
                return {it->id, s[id].id};
            q.insert(s[id]);
        } else {
            auto it = q.lower_bound(s[id]), next = it, prev =
                it;
            if (it != q.begin() && it != --q.end()) {
                ++next, --prev;
```

```
                if (intersect(*next, *prev))
                    return {next->id, prev->id};
            }
            q.erase(it);
        }
    }
    return {-1, -1};
}

// usage example
int main() {}
```

LiChaoTree.cpp

Description: desc

30d496, 73 lines

```
// https://cp-algorithms.com/geometry/convex_hull_trick.html

using T = long long;

struct Line {
    T a, d;

    T eval(T x) { return a * x + d; }
};

struct Node {
    Line line;
    Node *left = nullptr;
    Node *right = nullptr;

    Node(Line line) : line(line) {}

    void add_line(Line nline, T l, T r) {
        T m = (l + r) / 2;
        bool left_smaller = nline.eval(l) < line.eval(l);
        bool mid_smaller = nline.eval(m) < line.eval(m);
        if (mid_smaller) {
            swap(line, nline);
        }
        if (r - l == 1) {
            return;
        }
        if (left_smaller != mid_smaller) {
            if (left == nullptr)
                left = new Node(nline);
            else
                left->add_line(nline, l, m);
        } else {
            if (right == nullptr)
                right = new Node(nline);
            else
                right->add_line(nline, m, r);
        }
    }

    T get_min(T x, T l, T r) {
        if (r - l > 1) {
            T m = (l + r) / 2;
            if (x < m && left != nullptr) {
                return min(line.eval(x), left->get_min(x, l, m)
                    );
            }
            if (x >= m && right != nullptr) {
                return min(line.eval(x), right->get_min(x, m, r
                    ));
            }
        }
        return line.eval(x);
    }
};

struct LiChaoTree {
    T minx;
    T maxx;
    Node *root;

    LiChaoTree(T minx, T maxx) : minx(minx), maxx(maxx) { root
        = new Node({0, numeric_limits<T>::max() / 2}); }

    void add_line(Line line) { root->add_line(line, minx, maxx
        + 1); }

    T get_min(T x) { return root->get_min(x, minx, maxx + 1); }
};

// usage example
int main() {
    LiChaoTree t = LiChaoTree(1, 1e9);
    t.add_line({1, 3});
    t.add_line({2, 1});
    cout << t.get_min(1) << endl;
}
```

PointClassification.cpp

Description: desc

2866f2, 30 lines

```
using ll = long long;

enum class Position { Left, Right, Behind, Beyond, Origin,
    Destionation, Between };

// Classifies position of point p against vector a
Position classify(ll px, ll py, ll ax, ll ay) {
    ll cross = px * ay - py * ay;
    if (cross > 0) {
        return Position::Left;
    }
    if (cross < 0) {
        return Position::Right;
    }
    if (px == 0 && py == 0) {
        return Position::Origin;
    }
    if (px == ax && py == ay) {
        return Position::Destionation;
    }
    if (ax * px < 0 || ay * py < 0) {
        return Position::Beyond;
    }
    if (ax * ax + ay * ay < px * px + py * py) {
        return Position::Behind;
    }
    return Position::Between;
}

// usage example
int main() {}
```

PointInPolygon.cpp

Description: desc

ce59b6, 27 lines

```
using ll = long long;

int pointInPolygon(int qx, int qy, const vector<int> &x, const
    vector<int> &y) {
    int n = x.size();
    int cnt = 0;
    for (int i = 0, j = n - 1; i < n; j = i++) {
        if (y[i] == qy && (x[i] == qx || (y[j] == qy && (x[i]
            <= qx || x[j] <= qx) && (x[i] >= qx || x[j] >= qx)
            )))
            return 0;  // boundary
        if ((y[i] > qy) != (y[j] > qy)) {
            ll det = ((ll)x[i] - qx) * ((ll)y[j] - qy) - ((ll)x
                [j] - qx) * ((ll)y[i] - qy);
            if (det == 0)
                return 0;  // boundary
            if ((det > 0) != (y[j] > y[i]))
                ++cnt;
        }
    }
    return cnt % 2 == 0 ? -1 /* exterior */ : 1 /* interior */;
}

// usage example
int main() {
    vector<int> x{0, 0, 2, 2};
    vector<int> y{0, 2, 2, 0};
    cout << (1 == pointInPolygon(1, 1, x, y)) << endl;
    cout << (0 == pointInPolygon(0, 0, x, y)) << endl;
    cout << (-1 == pointInPolygon(0, 3, x, y)) << endl;
}
```

PointToSegmentDistance.cpp

Description: desc

467780, 31 lines

```
using ll = long long;

double fastHypot(double x, double y) {
    return sqrt(x * x + y * y);
}

double point_to_segment_distance(int x, int y, int x1, int y1,
    int x2, int y2) {
    ll dx = x2 - x1;
    ll dy = y2 - y1;
    ll px = x - x1;
    ll py = y - y1;
    ll squaredLength = dx * dx + dy * dy;
    ll dotProduct = dx * px + dy * py;
    if (dotProduct <= 0 || squaredLength == 0)
        return fastHypot(px, py);
    if (dotProduct >= squaredLength)
        return fastHypot(px - dx, py - dy);
    double q = (double)dotProduct / squaredLength;
    return fastHypot(px - q * dx, py - q * dy);
}

double point_to_line_distance(ll x, ll y, ll a, ll b, ll c) {
    return abs(a * x + b * y + c) / fastHypot(a, b);
}

// usage example
int main() {
    cout << fixed << setprecision(10);
    cout << point_to_segment_distance(0, 0, 0, 1, 1, 0) << endl
        ;
    cout << point_to_line_distance(0, 0, 1, 1, 1) << endl;
}
```

SegmentsIntersection.cpp

Description: desc

<optional>

a14bb2, 45 lines

```
using ll = long long;

bool is_cross_intersect(ll x1, ll y1, ll x2, ll y2, ll x3, ll
    y3, ll x4, ll y4) {
```

```
    ll z1 = (x2 - x1) * (y3 - y1) - (y2 - y1) * (x3 - x1);
    ll z2 = (x2 - x1) * (y4 - y1) - (y2 - y1) * (x4 - x1);
    ll z3 = (x4 - x3) * (y1 - y3) - (y4 - y3) * (x1 - x3);
    ll z4 = (x4 - x3) * (y2 - y3) - (y4 - y3) * (x2 - x3);
    return (z1 < 0 || z2 < 0) && (z1 > 0 || z2 > 0) && (z3 < 0
        || z4 < 0) && (z3 > 0 || z4 > 0);
}

bool is_cross_or_touch_intersect(ll x1, ll y1, ll x2, ll y2, ll
     x3, ll y3, ll x4, ll y4) {
    if (max(x1, x2) < min(x3, x4) || max(x3, x4) < min(x1, x2)
        || max(y1, y2) < min(y3, y4) ||
       max(y3, y4) < min(y1, y2))
        return false;
    ll z1 = (x2 - x1) * (y3 - y1) - (y2 - y1) * (x3 - x1);
    ll z2 = (x2 - x1) * (y4 - y1) - (y2 - y1) * (x4 - x1);
    ll z3 = (x4 - x3) * (y1 - y3) - (y4 - y3) * (x1 - x3);
    ll z4 = (x4 - x3) * (y2 - y3) - (y4 - y3) * (x2 - x3);
    return (z1 <= 0 || z2 <= 0) && (z1 >= 0 || z2 >= 0) && (z3
        <= 0 || z4 <= 0) && (z3 >= 0 || z4 >= 0);
}

optional<pair<double, double>> get_lines_intersection(ll x1, ll
     y1, ll x2, ll y2, ll x3, ll y3, ll x4, ll y4) {
    ll a1 = y2 - y1;
    ll b1 = x1 - x2;
    ll c1 = -(x1 * y2 - x2 * y1);
    ll a2 = y4 - y3;
    ll b2 = x3 - x4;
    ll c2 = -(x3 * y4 - x4 * y3);
    ll det = a1 * b2 - a2 * b1;
    if (det == 0)
        return {};
    double x = -(c1 * b2 - c2 * b1) / (double)det;
    double y = -(a1 * c2 - a2 * c1) / (double)det;
    return optional{make_pair(x, y)};
}

// usage example
int main() {
    optional<pair<double, double>> intersection =
        get_lines_intersection(0, 0, 4, 2, 2, -1, 2, 5);
    cout << (bool)intersection << endl;
    cout << intersection->first << " " << intersection->second
        << endl;

    optional<pair<double, double>> no_intersection =
        get_lines_intersection(0, 0, 0, 1, 1, 0, 1, 1);
    cout << (bool)no_intersection << endl;
}
```