2-SAT

```cpp
// 2 SAT
// Complexity O(n)
// Tested with: https://cses.fi/problemset/task/1684/

#include <bits/stdc++.h>
#define fi first
#define se second
#define mp make_pair
#define ALL(x) x.begin(), x.end()
#define bit(x) (1LL << (x))
#define getbit(x, i) (((x) >> (i)) & 1)
#define pb push_back
using namespace std;

mt19937_64 rd(chrono::steady_clock::now().time_since_epoch().count
    ());

template <typename T1, typename T2> bool mini(T1 &a, T2 b) {
    if (a > b) {a = b; return true;} return false;
}

template <typename T1, typename T2> bool maxi(T1 &a, T2 b) {
    if (a < b) {a = b; return true;} return false;
}

const double pi = acos(-1);
const int oo = 1e9;
const long long ooo = 1e18;

struct SAT {
    int n, cnt;
    vector <vector <int>> adj, revadj;
    vector <int> order, vis, comp;

    SAT (int _n) {
        n = 2 * _n;
        adj.resize(n);
        revadj.resize(n);
        vis = comp = vector <int> (n, 0);
        order.clear();
    }

    void addEdge(int a, int b) {
      adj[a ^ 1].push_back(b);
        adj[b ^ 1].push_back(a);
        revadj[a].push_back(b ^ 1);
        revadj[b].push_back(a ^ 1);
    }

    void dfs1(int u) {
        vis[u] = true;
        for (int v : adj[u])
            if (!vis[v])
                dfs1(v);
```

```
        order.push_back(u);
    }

    void dfs2(int u) {
        vis[u] = true;
        for (int v : revadj[u])
            if (!vis[v])
                dfs2(v);
        comp[u] = cnt;
    }

    vector <int> solve() {
        for (int i = 0; i < n; i++)
            if (!vis[i])
                dfs1(i);

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

        reverse(order.begin(), order.end());

        for (int i : order)
            if (!vis[i])
                cnt++, dfs2(i);

        for (int i = 0; i < n; i += 2)
            if (comp[i] == comp[i ^ 1])
                return vector <int> (1, -1);

        vector <int> trueValues;
        for (int i = 0; i < n; i += 2)
            if (comp[i] < comp[i ^ 1])
                trueValues.push_back(i >> 1);
        return trueValues;
    }

};
```

Bridge

```cpp
#include <bits/stdc++.h>
using namespace std;
const int lim = 1e5 + 5;
bool vis[lim];
int disc[lim], low[lim];
vector<vector<int>> component;
vector<int> s, edges[lim];
int t = 0;
void bridge(int nd, int prev = -1) {
    vis[nd] = 1;
    low[nd] = disc[nd] = ++t;
    for(auto i : edges[nd]) {
        if(i == prev)
            continue;
        else if(!vis[i]) {
            bridge(i, nd);
            low[nd] = min(low[nd], low[i]);
            if(low[i] > disc[nd]) {
                // the edge nd to i is a bridge

            }
        }
        else {
            low[nd] = min(low[nd], disc[i]);
        }
    }

}
```

Tarjan SCC

```cpp
#include <bits/stdc++.h>
using namespace std;
const int lim = 1e5 + 5;
bool vis[lim], active[lim];
int disc[lim], low[lim];
vector<vector<int>> component;
vector<int> s, edges[lim];
int t = 0;
void tarjan(int nd) {
    vis[nd] = 1;
    low[nd] = disc[nd] = ++t;
    active[nd] = 1;
    s.push_back(nd);
    for(auto i : edges[nd]) {
        if(!vis[i]) {
            tarjan(i);
            low[nd] = min(low[nd], low[i]);
        }
        else if(active[i]) {
            low[nd] = min(low[nd], disc[i]);
        }
    }
    if(disc[nd] == low[nd]) {
        vector<int> new_component;
        do {
            new_component.push_back(s.back());
            active[s.back()] = 0;
            s.pop_back();
        } while(new_component.back() != nd);
        component.push_back(new_component);
    }
}
```

Convex Hull

```cpp
// header file
#include <bits/stdc++.h>
#define endl "\n"
#define ll long long
#define pb push_back
#define fi first
#define se second
using namespace std;
ll cross_product(pair<ll, ll> a, pair<ll, ll> b, pair<ll, ll> c) {
    b.fi -= a.fi;
    c.fi -= a.fi;
    b.se -= a.se;
    c.se -= a.se;
    return b.fi * c.se - c.fi * b.se;
}
vector<pair<ll, ll>> convex_hull(vector<pair<ll, ll>> a) {
  int n = a.size();
  sort(a.begin(), a.end());
  vector<pair<ll, ll>> s;
  s.pb(a[0]);
  s.pb(a[1]);
  for(int i = 2; i < n; ++i) {
    while(s.size() > 1 && cross_product(s[s.size() - 2], s[s.size()
     - 1], a[i]) < 0)
      s.pop_back();
    s.pb(a[i]);
  }
  vector<pair<ll, ll>> ans;
  for(auto i : s)
    ans.pb(i);
  ans.pop_back();
  s.clear();
  s.pb(a[n - 1]);
  s.pb(a[n - 2]);
  for(int i = n - 3; i >= 0; --i) {
    while(s.size() > 1 && cross_product(s[s.size() - 2], s[s.size()
     - 1], a[i]) < 0)
      s.pop_back();
    s.pb(a[i]);
  }
  for(auto i : s)
    ans.pb(i);
  ans.pop_back();
  return ans;
}
```

Line Segment Intersection

```cpp
#include <bits/stdc++.h>
#define ll long long
#define fi first
#define se second
using namespace std;
ll cross(pair<ll, ll> a, pair<ll, ll> b) {
    return a.first * b.second - b.first * a.second;
}
pair<ll, ll> sub(pair<ll, ll> a, pair<ll, ll> b) {
    return {a.fi - b.fi, a.se - b.se};
}
bool intersect(pair<pair<ll, ll>, pair<ll, ll>> a, pair<pair<ll, ll>, pair<ll, ll>> b) {
    ll ac = cross(sub(a.se, a.fi), sub(b.fi, a.fi)), ad = cross(sub(a.se, a.fi), sub(b.se, a.fi))
    , bc = cross(sub(a.fi, a.se), sub(b.fi, a.se)), bd = cross(sub(a.fi, a.se), sub(b.se, a.se));
    if(ac == 0 && ad == 0 && bc == 0 && bd == 0) {
        if((a.fi >= min(b.fi, b.se) && a.fi <= max(b.fi, b.se)) || (a.se >= min(b.fi, b.se) && a.se <= max(b.fi, b.se))
        || (b.fi >= min(a.fi, a.se) && b.fi <= max(a.fi, a.se)) || (b.se >= min(a.fi, a.se) && b.se <= max(a.fi, a.se)))
            return 1;
        else
            return 0;
    }
    ll ca = cross(sub(b.se, b.fi), sub(a.fi, b.fi)), cb = cross(sub(b.se, b.fi), sub(a.se, b.fi))
    , da = cross(sub(b.fi, b.se), sub(a.fi, b.se)), db = cross(sub(b.fi, b.se), sub(a.se, b.se));
    bool ans1 = 0, ans2 = 0;
    if(ac <= 0 && ad >= 0 && bc >= 0 && bd <= 0) {
        ans1 = 1;
    }
    if(ac >= 0 && ad <= 0 && bc <= 0 && bd >= 0) {
        ans1 = 1;
    }
    if(ca <= 0 && cb >= 0 && da >= 0 && db <= 0) {
        ans2 = 1;
    }
    if(ca >= 0 && cb <= 0 && da <= 0 && db >= 0) {
        ans2 = 1;
    }
    return ans1 && ans2;
}
```

Dinic

```cpp
// Network Flow
// Complexity O(n^3 log) in worst case if binary search
// Still do pretty well without binary search
// Tested with: https://cses.fi/problemset/task/1695/
#include <bits/stdc++.h>

using namespace std;

struct Dinic {
    struct Edge {
    int v,pos,flow,cap;

    Edge(){}
    Edge(int _v, int _pos, int _flow, int _cap) {
      v = _v;
      pos = _pos;
      flow = _flow;
      cap = _cap;
    }
  };

  int n;
  vector <vector<Edge>> adj;
  vector <int> ptr,dis;

  Dinic(int _n) {
    n = _n;
    adj.resize(n + 5);
    ptr = dis = vector <int> (n + 5, 0);
  }

  void addEdge(int u, int v, int cap, int rcap = 0) {
    Edge a = Edge(v, adj[v].size(), 0, cap);
    Edge b = Edge(u, adj[u].size(), 0, rcap);
    adj[u].push_back(a);
    adj[v].push_back(b);
  }

  bool bfs(int s, int t) {
    ptr = dis = vector <int>(n + 5);

    dis[s] = 1;
    queue <int> q({s});
    while (q.size()) {
      int u = q.front(); q.pop();

      for (Edge e : adj[u]) if (dis[e.v] == 0 && e.flow < e.cap) {
        dis[e.v] = dis[u] + 1;
        q.push(e.v);
      }
    }
    return dis[t] > 0;
  }

```

```
  int dfs(int u, int t, int flow) {
    if (u == t)
      return flow;

    for (int &i = ptr[u]; i < (int) adj[u].size(); i++) {
      Edge &e = adj[u][i];
      int dif = e.cap - e.flow;
      if (dis[u] + 1 != dis[e.v] || dif == 0)
        continue;

      if (int val = dfs(e.v, t, min(flow, dif)) > 0) {
        e.flow += val; adj[e.v][e.pos].flow -= val;
        return val;
      }
    }
    return 0;
  }

  void mincut(int s, int t) {
    int res = 0;
    while (bfs(s, t))
      while (int val = dfs(s, t, (int) 1e9))
        res += val;
    cout << res << "\n";
    bfs(s, t);
    for (int i = 1; i <= n; i++)
    for (Edge e : adj[i])
      if (dis[i] > 0 && dis[e.v] == 0)
        cout << i << " " << e.v << "\n";
  }
};
```

Max Matching

```cpp
// Matching on Bipartite Graph
// Complexity: O(n sqrt n)
// Tested with: https://judge.yosupo.jp/problem/bipartitematching
// Note: to achieve "arbitrary" (not maximum) flow with lower bound :
// - Create a new graph with a new source s' and a new sink t'
// - if there is an edge from u => v with cap(u, v) >= dem(u, v)
// - cap(s', v) += dem(u, v)
// - cap(u, t') += dem(u, v)
// - cap(t, s) += oo
// - cap(u, v) += cap(u, v) - dem(u, v)
#include <bits/stdc++.h>

using namespace std;
const int N = 1e5 + 5;
const int oo = 1e9;

struct HopCroft {
  int n,m,t;
  vector <vector <int>> adj;
  vector <int> vis, dis, match, matched;

  HopCroft(int _n, int _m) {
    n = _n;
    m = _m;
    adj.resize(n + 5);
        vis = dis = matched = vector <int> (n + 5, 0);
        match.assign(m + 5, -1);
  }

  void add(int u, int v) {
    adj[u].push_back(v);
  }

  bool bfs() {
    for (int i = 0; i < n; i++)
      dis[i] = oo;

    queue <int> q;

    for (int i = 0; i < n; i++)
      if (!matched[i])
        q.push(i), dis[i] = 0;

    bool ok = false;
    while (q.size()) {
      int u = q.front(); q.pop();

      for (int v : adj[u]) {
        if (match[v] < 0)
          ok = true;
        else if (dis[match[v]] == oo) {
          dis[match[v]] = dis[u] + 1;
          q.push(match[v]);
```

```
        }
      }
    }

    return ok;
  }

  bool dfs(int u) {
    if (vis[u] == t)
      return false;
    vis[u] = t;
    for (int v : adj[u]) if (match[v] == -1) {
      match[v] = u;
      matched[u] = true;
      return true;
    }

    for (int v : adj[u]) if (match[v] != -1 && dis[match[v]] == dis
    [u] + 1 && dfs(match[v]))  {
      match[v] = u;
      matched[u] = true;
      return true;
    }
    return false;
  }

  void maxmatch() {
    int res = 0;
    t = 0;
    while (bfs()) {
      t++;
      for (int i = 0; i < n; i++) {
        if (!matched[i] && dfs(i))
          res++;
      }
    }
    // cout << res << "\n";

    // for (int  i = 0; i < m; i++)
    //  if (match[i] >= 0)
    //    cout << match[i] << " " << i << "\n";
  }
};
```

MCMF

```cpp
// Min cost Max Flow
// Tested with: https://codeforces.com/contest/237/problem/E

#include <bits/stdc++.h>
#define fi first
#define se second
#define mp make_pair
using namespace std;
template <typename T1, typename T2> bool mini(T1 &a, T2 b) {
  if (a > b) {a = b; return true;} return false;
}
template <typename T1, typename T2> bool maxi(T1 &a, T2 b) {
  if (a < b) {a = b; return true;} return false;
}
const int N = 1e5 + 5;
const int oo = 1e9;

struct MCMF {
  struct edge{
    int u,v, flow, cap, cost;
    edge(int _u, int _v, int _cap, int _cost) {
      u = _u;
      v = _v;
      cap = _cap;
      cost = _cost;
      flow = 0;
    }

    int rem() {
      return cap - flow;
    }
  };

  int n;
  vector <bool> inq;
  vector <int> ptr;
  vector <int> dis;
  vector <vector <int>> adj;
  vector <edge> g;

  void init(int _n) {
    n = _n;
    ptr.resize(n + 5);
    adj.resize(n + 5);
    inq.resize(n + 5);
    dis.resize(n + 5);
  }

  void addEdge(int u, int v, int cap, int cost) {
    adj[u].push_back(g.size());
    g.push_back(edge(u, v, cap,cost));
    adj[v].push_back(g.size());
    g.push_back(edge(v, u, 0, -cost));
  }
```

```
55
56    bool path(int s, int t) {
57      for (int i = 1; i <= n; i++) {
58        dis[i] = oo;
59        inq[i] = false;
60        ptr[i] = -1;
61      }
62
63      queue <int> q({s});
64      dis[s] = 0; inq[s] = true;
65      while (q.size()) {
66        int u = q.front(); q.pop(); inq[u] = false;
67        for (int i : adj[u]) if (g[i].rem() > 0) {
68          if (mini(dis[g[i].v], dis[u] + g[i].cost)) {
69            if (!inq[g[i].v]) {
70              q.push(g[i].v);
71              inq[g[i].v] = true;
72            }
73            ptr[g[i].v] = i;
74          }
75        }
76      }
77      return ptr[t] != -1;
78    }
79
80    pair <int, int> mcmf(int s, int t) {
81      int totflow = 0, totcost = 0;
82      while (path(s, t)) {
83        int flow = oo;
84        for (int u = t; u != s; u = g[ptr[u]].u)
85          mini(flow, g[ptr[u]].rem());
86        totflow += flow;
87        totcost += flow * dis[t];
88        for (int u = t; u != s; u = g[ptr[u]].u) {
89          g[ptr[u]].flow += flow;
90          g[ptr[u] ^ 1].flow -= flow;
91        }
92      }
93      return mp(totflow, totcost);
94    }
95  };
```

BIT 2D

```cpp
// FenwickTree 2D
// Complexity: O(n log^2 n)

#include <bits/stdc++.h>
#define mp make_pair
#define fi first
#define se second
#define pb push_back
#define bit(n) (1LL << (n))
#define getbit(x, i) (((x) >> (i)) & 1)
#define pii pair<int, int>
#define ALL(x) x.begin(), x.end()
using namespace std;
const int M = 5e5 + 5;
const int N = 2e5 + 5;
const int mod = 1e9 + 7;
const int oo = 1e9;
const long long ooo = 1e18;

const double pi = acos(-1);
template<typename T1, typename T2> bool mini(T1 &a, T2 b) {if(a > b) a = b; else return 0; return 1;}
template<typename T1, typename T2> bool maxi(T1 &a, T2 b) {if(a < b) a = b; else return 0; return 1;}

typedef long long ll;

struct BIT2D {
    int n;
    vector <vector <int>> mx, yval;

    BIT2D(int n) {
        this->n = n;
        mx.resize(n + 1);
        yval.resize(n + 1);
    }

    void fakeupdate(int x, int y) {
        for (; x <= n; x += x & -x)
            yval[x].push_back(y);
    }

    void init() {
        for (int i = 1; i <= n; i++) {
            mx[i].assign(yval[i].size() + 1, -oo);
            sort(ALL(yval[i]));
        }
    }

    void update(int x, int y, int val) {
        for (; x <= n; x += x & -x) {
            int i = upper_bound(ALL(yval[x]), y) - yval[x].begin();
            for (; i < (int) mx[x].size(); i += i & -i) {
                maxi(mx[x][i], val);
```

```
            }
        }
    }

    int getmax(int x, int y) {
        int res = -oo;
        for (; x; x -= x & -x) {
            int i = upper_bound(ALL(yval[x]), y) - yval[x].begin();
            for (; i; i -= i & -i) {
                maxi(res, mx[x][i]);
            }
        }
        return res;
    }
};
```

Dynamic CHT

```cpp
#include <bits/stdc++.h>
#define ll long long
// source: https://github.com/niklasb/contest-algos/blob/master/convex_hull/dynamic.cpp
// Used in problem CS Squared Ends
// Problem: A is an array of n integers. The cost of subarray A[l...r] is (A[l]-A[r])^2. Partition
// the array into K subarrays having a minimum total cost
// In case of initializing 'ans', check if 1e18 is enough. Might need LLONG_MAX

using namespace std;
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will maintain upper hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;

        // **** May need long double typecasting here
        return (long double)(x->b - y->b)*(z->m - y->m) >= (long double)(y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

Persistent Dynamic Segment Tree

```cpp
#include <bits/stdc++.h>
using namespace std;
struct node {
    int val = 0;
    node *l = nullptr, *r = nullptr;
    node() {

    }
    node(node *le, node *re) {
        l = le, r = re;
        if(l)
            val += l->val;
        if(r);
            val += r->val;
    }
};
struct segment_tree {
    vector<node*> a = {new node()};
    node *empty = new node();
    int size = 1e9 + 1e5;
    void update(node *nd, node *prev, int cl, int cr, int idx, int
    val) {
        int mid = (cl + cr) / 2;
        if(cl == cr) {
            nd->val = prev->val + val;
        }
        else if(idx <= mid) {
            nd->l = new node();
            nd->r = prev->r;
            if(prev->l != nullptr)
                update(nd->l, prev->l, cl, mid, idx, val);
            else
                update(nd->l, empty, cl, mid, idx, val);
            if(nd->l)
                nd->val += nd->l->val;
            if(nd->r)
                nd->val += nd->r->val;
        }
        else {
            nd->l = prev->l;
            nd->r = new node();
            if(prev->r != nullptr)
                update(nd->r, prev->r, mid + 1, cr, idx, val);
            else
                update(nd->r, empty, mid + 1, cr, idx, val);
            if(nd->l)
                nd->val += nd->l->val;
            if(nd->r)
                nd->val += nd->r->val;
        }
    }
    void update(int idx, int val) {
        a.push_back(new node());
        update(a[a.size() - 1], a[a.size() - 2], 1, size, idx, val)
```

```
    ;
    }
    int query(node *l, node *r, int cl, int cr, int k) {
        int mid = (cl + cr) / 2;
        int tmpr = 0, tmpl = 0;
        if(cl == cr)
            return cl;
        if(r->l != nullptr)
            tmpr = r->l->val;
        if(l->l != nullptr)
            tmpl = l->l->val;
        // count di kiri ga sampe k
        // berarti k nya di kanan
        if(tmpr - tmpl < k) {
            if(l->r != nullptr)
                return query(l->r, r->r, mid + 1, cr, k - (tmpr -
    tmpl));
            else
                return query(empty, r->r, mid + 1, cr, k - (tmpr -
    tmpl));
        }
        else {
            if(l->l != nullptr)
                return query(l->l, r->l, cl, mid, k);
            else
                return query(empty, r->l, cl, mid, k);
        }
    }
    int query(int l, int r, int k) {
        return query(a[l - 1], a[r], 1, size, k);
    }
};
```

String Hash

```cpp
#include <bits/stdc++.h>
using namespace std;
struct string_hash {
    int lim, mod_sz, key;
    long long power[mod_sz][lim];
    vector<int> val[mod_sz];
    vector<int> mod;
    string str;
    void build(int Lim, int Mod_sz, int Key, vector<int> Mod) {
        lim = Lim, mod_sz = Mod_sz, key = Key, mod = Mod;
        long long tmp[mod_sz];
        memset(tmp, 0, sizeof(tmp));
        for(int i = 0; i < str.size(); ++i) {
            for(int j = 0; j < mod_sz; ++j) {
                tmp[j] = tmp[j] * key + str[i];
                tmp[j] %= mod[j];
                val[j].push_back(tmp[j]);
            }
        }
    }
    vector<int> q(int l, int r) {
        vector<int> ret;
        if(l == 0) {
            for(int i = 0; i < mod_sz; ++i)
                ret.push_back(val[i][r]);
        }
        else {
            for(int i = 0; i < mod_sz; ++i) {
                ret.push_back(val[i][r] - ((1ll * val[i][l - 1] *
    power[i][r - l + 1]) % mod[i]));
                if(ret.back() < 0)
                    ret.back() += mod[i];
            }
        }
        return ret;
    }
};
```

Treap

```cpp
#include <bits/stdc++.h>
using namespace std;
struct string_hash {
    int lim, mod_sz, key;
    long long power[mod_sz][lim];
    vector<int> val[mod_sz];
    vector<int> mod;
    string str;
    void build(int Lim, int Mod_sz, int Key, vector<int> Mod) {
        lim = Lim, mod_sz = Mod_sz, key = Key, mod = Mod;
        long long tmp[mod_sz];
        memset(tmp, 0, sizeof(tmp));
        for(int i = 0; i < str.size(); ++i) {
            for(int j = 0; j < mod_sz; ++j) {
                tmp[j] = tmp[j] * key + str[i];
                tmp[j] %= mod[j];
                val[j].push_back(tmp[j]);
            }
        }
    }
    vector<int> q(int l, int r) {
        vector<int> ret;
        if(l == 0) {
            for(int i = 0; i < mod_sz; ++i)
                ret.push_back(val[i][r]);
        }
        else {
            for(int i = 0; i < mod_sz; ++i) {
                ret.push_back(val[i][r] - ((1ll * val[i][l - 1] *
    power[i][r - l + 1]) % mod[i]));
                if(ret.back() < 0)
                    ret.back() += mod[i];
            }
        }
        return ret;
    }
};
```

Operasi Lain-lain

- Memasukkan elemen X pada posisi K
  1. Split(T, K-1, L, R)
  2. Merge(temp, L, X)
  3. Merge(T, temp, R)
- Menghapus elemen pada posisi K
  1. Split(T, K, L, R)
  2. Split(L, K-1, L1, L2)
  3. Merge(T, L1, R)
- Query sum [X, Y]
  1. Split(T, Y, L, R)
  2. Split(L, X-1, L1, L2)
  3. cout<<L2.sum<<"\n";
  4. Merge(L, L1, L2)
  5. Merge(T, L, R)

- Menambah V pada posisi K.
  1. Split(T, K, L, R)
  2. Split(T, K-1, L1, L2)
  3. L2.val+=V
  4. Merge(L, L1, L2)
  5. Merge(T, L, R)

PBDS

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less_equal<int>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_multiset;
typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;
ordered_set s;
s.insert(10);
s.insert(30);
s.insert(40);
s.insert(50);
s.insert(35);
s.order_of_key(30); // 1
s.order_of_key(36); // 3
*s.find_by_order(2); // 35
*s.find_by_order(0); // 10
const int RANDOM = chrono::high_resolution_clock::now().
    time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<key, int, chash> table;
```

Matrix Mod Expo

```cpp
template<typename T> struct Matrix {
  int sz;
  vector<vector<T>> val;

  Matrix(int _sz, bool isIdentity = false) {
    sz = _sz;
    val.assign(_sz, vector<T>(_sz, 0));
    if (isIdentity) {
      for (int i = 0; i < _sz; i++) {
        val[i][i] = 1;
      }
    }
  }

  Matrix operator * (const Matrix &other) const {
    Matrix ret = Matrix(sz);
    for (int i = 0; i < sz; i++) {
      for (int j = 0; j < sz; j++) {
        for (int k = 0; k < sz; k++) {
          maddto<T>(ret.val[i][j], mmul<T>(val[i][k], other.val[k][j], mod), mod);
        }
      }
    }
    return ret;
  }

  void print() {
    for (int i = 0; i < sz; i++) {
      for (int j = 0; j < sz; j++) {
        cout << val[i][j] << " ";
      }
      cout << '\n';
    }
  }
};

template<typename T> Matrix<T> matexpo(Matrix<T> x, ll y) {
  if (!y) {return Matrix<T>(x.sz, 1);}

  Matrix<T> t = matexpo(x, y >> 1);
  t = t * t;
  if (y & 1ll) {t = t * x;}
  return t;
}
```

Operator Overloading

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Edge {
  int a, b, w;
  bool operator<(const Edge &y) const { return w < y.w; }
};

int main() {
  int M = 4;
  set<Edge> v;
  for (int i = 0; i < M; ++i) {
    int a, b, w;
    cin >> a >> b >> w;
    v.insert({a, b, w});
  }
  for (Edge e : v) cout << e.a << " " << e.b << " " << e.w << "\n";
}
```

Li Chao Tree

```cpp
struct Line {
  ll m, c;

  Line(ll m, ll c) : m(m), c(c) {}

  ll operator() (ll x) {
    return 1ll * m * x + 1ll * c;
  }

  ld intersect(Line other) {
    return (ld) (c - other.c) / (ld) (other.m - m);
  }
};

struct LiChaoTree {
  int sz;
  vector<Line> seg;

  LiChaoTree(int sz) : sz(sz) {
    seg.assign((sz + 2) << 2, Line(0, INF));
  }

  // Important: Nodes are of the form [L, R) instead of [L, R]

  void insert(int pos, int l, int r, Line newLine) {
    int mid = (l + r) >> 1;
    bool lless = newLine(l) < seg[pos](l);
    bool mless = newLine(mid) < seg[pos](mid);

    if (mless) {swap(newLine, seg[pos]);}

    if (r == l + 1) {return;}

    int lc = pos << 1, rc = lc | 1;
    if (lless ^ mless) {insert(lc, l, mid, newLine);}
    else {insert(rc, mid, r, newLine);}
  }
  void insert(Line newLine) {insert(1, 1, sz, newLine);}

  ll query(int pos, int l, int r, int idx) {
    if (r == l + 1) {return seg[pos](idx);}

    int mid = (l + r) >> 1, lc = pos << 1, rc = lc | 1;
    if (idx < mid) {return min(seg[pos](idx), query(lc, l, mid, idx
    ));}
    else {return min(seg[pos](idx), query(rc, mid, r, idx));}
  }
  ll query(int idx) {return query(1, 1, sz, idx);}
};
```

Dynamic Li Chao Tree

```
struct Line {
  ll m, c;

  Line(ll m, ll c) : m(m), c(c) {}

  ll operator() (ll x) {
    return 1ll * m * x + 1ll * c;
  }

  ld intersect(Line other) {
    return (ld) (c - other.c) / (ld) (other.m - m);
  }
};

struct DynamicLiChaoTree {
  struct Node {
    Line line;
    Node *left, *right;

    Node() {}
    Node(Line L) {
      line = L;
      left = right = nullptr;
    }
  };

  ll boundL, boundR;
  Node *root;

  DynamicLiChaoTree() {}
  DynamicLiChaoTree(ll boundL, ll boundR) :
    boundL(boundL), boundR(boundR), root(new Node(Line(0, INF)))
  {}

  // Important: Nodes are of the form [L, R) instead of [L, R]

  void insert(Node *cur, ll l, ll r, Line newLine) {
    ll mid = l + (r - l) / 2;
    bool lless = newLine(l) < cur->line(l);
    bool mless = newLine(mid) < cur->line(mid);

    if (mless) {swap(newLine, cur->line);}

    if (r == l + 1) {return;}

    if (lless ^ mless) {
      if (!cur->left) cur->left = new Node(Line(0, INF));
      insert(cur->left, l, mid, newLine);
    } else {
      if (!cur->right) cur->right = new Node(Line(0, INF));
      insert(cur->right, mid, r, newLine);
    }
  }
  void insert(Line newLine) {insert(root, boundL, boundR, newLine)
```

```
    ;}

  ll query(Node *cur, ll l, ll r, ll idx) {
    ll ret = cur->line(idx);

    if (r == l + 1) {return ret;}

    ll mid = l + (r - l) / 2;
    if (idx < mid) {
      if (cur->left) ret = min(ret, query(cur->left, l, mid, idx));
    } else {
      if (cur->right) ret = min(ret, query(cur->right, mid, r, idx)
    );
    }
    return ret;
  }
  ll query(ll idx) {return query(root, boundL, boundR, idx);}
};
```

HLD + LCA

```
segment_tree seg;
void dfs(int nd) {
    vis[nd] = 1;
    subtree[nd] = 1;
    for(int i = 1; i < 17; ++i) {
        par[i][nd] = par[i - 1][par[i - 1][nd]];
    }
    for(auto i : edges[nd]) {
        if(!vis[i]) {
            depth[i] = depth[nd] + 1;
            par[0][i] = nd;
            dfs(i);
            child[nd].push_back(i);
            subtree[nd] += subtree[i];
        }
    }
    for(auto &i : child[nd])
        if(subtree[i] > subtree[child[nd][0]])
            swap(i, child[nd][0]);
}
int in[lim], ti, root[lim];
void dfs2(int nd) {
    in[nd] = ti++;
    for(auto i : child[nd]) {
        root[i] = (i == child[nd][0] ? root[nd] : i);
        dfs2(i);
    }
}
int lca(int u, int v) {
    if(depth[u] > depth[v]) {
        swap(u, v);
    }
    // depth v lebih besar
    for(int i = 16; i >= 0; --i) {
        if(depth[par[i][v]] >= depth[u])
            v = par[i][v];
    }
    if(u == v)
        return u;
    for(int i = 16; i >= 0; --i) {
        if(par[i][v] != par[i][u])
            u = par[i][u], v = par[i][v];
    }
    return par[0][u];
}
int query(int u, int v) {
    // u itu yg root
    if(depth[u] > depth[v])
        swap(u, v);
    int res = 0;
    while(root[v] != root[u]) {
        res = max(res, (int)seg.query(in[root[v]], in[v]));
        v = par[0][root[v]];
        if(depth[u] > depth[v])
```

```
            swap(u, v);
        }
        res = max(res, (int)seg.query(in[u], in[v]));
        return res;
}
// for update, use same as query, just change to update
```