

2-SAT

```
1 // 2 SAT
2 // Complexity O(n)
3 // Tested with: https://cses.fi/problemset/task/1684/
4
5 #include <bits/stdc++.h>
6 #define fi first
7 #define se second
8 #define mp make_pair
9 #define ALL(x) x.begin(), x.end()
10 #define bit(x) (1LL << (x))
11 #define getbit(x, i) (((x) >> (i)) & 1)
12 #define pb push_back
13 using namespace std;
14
15 mt19937_64 rd(chrono::steady_clock::now().time_since_epoch().count
16   ());
17
18 template <typename T1, typename T2> bool mini(T1 &a, T2 b) {
19     if (a > b) {a = b; return true;} return false;
20 }
21
22 template <typename T1, typename T2> bool maxi(T1 &a, T2 b) {
23     if (a < b) {a = b; return true;} return false;
24 }
25
26 const double pi = acos(-1);
27 const int oo = 1e9;
28 const long long ooo = 1e18;
29
30 struct SAT {
31     int n, cnt;
32     vector <vector <int>> adj, revadj;
33     vector <int> order, vis, comp;
34
35     SAT (int _n) {
36         n = 2 * _n;
37         adj.resize(n);
38         revadj.resize(n);
39         vis = comp = vector <int> (n, 0);
40         order.clear();
41     }
42
43     void addEdge(int a, int b) {
44         adj[a ^ 1].push_back(b);
45         adj[b ^ 1].push_back(a);
46         revadj[a].push_back(b ^ 1);
47         revadj[b].push_back(a ^ 1);
48     }
49
50     void dfs1(int u) {
51         vis[u] = true;
52         for (int v : adj[u])
53             if (!vis[v])
54                 dfs1(v);
55     }
56 }
```

```

54     order.push_back(u);
55 }
56
57 void dfs2(int u) {
58     vis[u] = true;
59     for (int v : revadj[u])
60         if (!vis[v])
61             dfs2(v);
62     comp[u] = cnt;
63 }
64
65 vector<int> solve() {
66     for (int i = 0; i < n; i++)
67         if (!vis[i])
68             dfs1(i);
69
70     for (int i = 0; i < n; i++)
71         vis[i] = 0;
72
73     reverse(order.begin(), order.end());
74
75     for (int i : order)
76         if (!vis[i])
77             cnt++, dfs2(i);
78
79     for (int i = 0; i < n; i += 2)
80         if (comp[i] == comp[i ^ 1])
81             return vector<int> (1, -1);
82
83     vector<int> trueValues;
84     for (int i = 0; i < n; i += 2)
85         if (comp[i] < comp[i ^ 1])
86             trueValues.push_back(i >> 1);
87     return trueValues;
88 }
89
90 };

```

Bridge

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 const int lim = 1e5 + 5;
4 bool vis[lim];
5 int disc[lim], low[lim];
6 vector<vector<int>> component;
7 vector<int> s, edges[lim];
8 int t = 0;
9 void bridge(int nd, int prev = -1) {
10     vis[nd] = 1;
11     low[nd] = disc[nd] = ++t;
12     for(auto i : edges[nd]) {
13         if(i == prev)
14             continue;
15         else if(!vis[i]) {
16             bridge(i, nd);
17             low[nd] = min(low[nd], low[i]);
18             if(low[i] > disc[nd]) {
19                 // the edge nd to i is a bridge
20             }
21         }
22     }
23     else {
24         low[nd] = min(low[nd], disc[i]);
25     }
26 }
27
28 }
```

Tarjan SCC

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 const int lim = 1e5 + 5;
4 bool vis[lim], active[lim];
5 int disc[lim], low[lim];
6 vector<vector<int>> component;
7 vector<int> s, edges[lim];
8 int t = 0;
9 void tarjan(int nd) {
10     vis[nd] = 1;
11     low[nd] = disc[nd] = ++t;
12     active[nd] = 1;
13     s.push_back(nd);
14     for(auto i : edges[nd]) {
15         if(!vis[i]) {
16             tarjan(i);
17             low[nd] = min(low[nd], low[i]);
18         }
19         else if(active[i]) {
20             low[nd] = min(low[nd], disc[i]);
21         }
22     }
23     if(disc[nd] == low[nd]) {
24         vector<int> new_component;
25         do {
26             new_component.push_back(s.back());
27             active[s.back()] = 0;
28             s.pop_back();
29         } while(new_component.back() != nd);
30         component.push_back(new_component);
31     }
32 }
```

Convex Hull

```
1 // header file
2 #include <bits/stdc++.h>
3 #define endl "\n"
4 #define ll long long
5 #define pb push_back
6 #define fi first
7 #define se second
8 using namespace std;
9 ll cross_product(pair<ll, ll> a, pair<ll, ll> b, pair<ll, ll> c) {
10     b.fi -= a.fi;
11     c.fi -= a.fi;
12     b.se -= a.se;
13     c.se -= a.se;
14     return b.fi * c.se - c.fi * b.se;
15 }
16 vector<pair<ll, ll>> convex_hull(vector<pair<ll, ll>> a) {
17     int n = a.size();
18     sort(a.begin(), a.end());
19     vector<pair<ll, ll>> s;
20     s.pb(a[0]);
21     s.pb(a[1]);
22     for(int i = 2; i < n; ++i) {
23         while(s.size() > 1 && cross_product(s[s.size() - 2], s[s.size()
24             - 1], a[i]) < 0)
25             s.pop_back();
26         s.pb(a[i]);
27     }
28     vector<pair<ll, ll>> ans;
29     for(auto i : s)
30         ans.pb(i);
31     ans.pop_back();
32     s.clear();
33     s.pb(a[n - 1]);
34     s.pb(a[n - 2]);
35     for(int i = n - 3; i >= 0; --i) {
36         while(s.size() > 1 && cross_product(s[s.size() - 2], s[s.size()
37             - 1], a[i]) < 0)
38             s.pop_back();
39         s.pb(a[i]);
40     }
41     for(auto i : s)
42         ans.pb(i);
43     ans.pop_back();
44     return ans;
45 }
```

Line Segment Intersection

```
1 #include <bits/stdc++.h>
2 #define ll long long
3 #define fi first
4 #define se second
5 using namespace std;
6 ll cross(pair<ll, ll> a, pair<ll, ll> b) {
7     return a.first * b.second - b.first * a.second;
8 }
9 pair<ll, ll> sub(pair<ll, ll> a, pair<ll, ll> b) {
10     return {a.fi - b.fi, a.se - b.se};
11 }
12 bool intersect(pair<pair<ll, ll>, pair<ll, ll>> a, pair<pair<ll, ll>
13 >, pair<ll, ll>> b) {
14     ll ac = cross(sub(a.se, a.fi), sub(b.fi, a.fi)), ad = cross(sub(
15     a.se, a.fi), sub(b.se, a.fi))
16     , bc = cross(sub(a.fi, a.se), sub(b.fi, a.se)), bd = cross(sub(
17     a.fi, a.se), sub(b.se, a.se));
18     if(ac == 0 && ad == 0 && bc == 0 && bd == 0) {
19         if((a.fi >= min(b.fi, b.se) && a.fi <= max(b.fi, b.se)) ||
20         (a.se >= min(b.fi, b.se) && a.se <= max(b.fi, b.se))
21         || (b.fi >= min(a.fi, a.se) && b.fi <= max(a.fi, a.se)) ||
22         (b.se >= min(a.fi, a.se) && b.se <= max(a.fi, a.se)))
23         return 1;
24     }
25     else
26         return 0;
27 }
28 ll ca = cross(sub(b.se, b.fi), sub(a.fi, b.fi)), cb = cross(sub(
29     b.se, b.fi), sub(a.se, b.fi))
30     , da = cross(sub(b.fi, b.se), sub(a.fi, b.se)), db = cross(sub(
31     b.fi, b.se), sub(a.se, b.se));
32 bool ans1 = 0, ans2 = 0;
33 if(ac <= 0 && ad >= 0 && bc >= 0 && bd <= 0) {
34     ans1 = 1;
35 }
36 if(ac >= 0 && ad <= 0 && bc <= 0 && bd >= 0) {
37     ans1 = 1;
38 }
39 if(ca <= 0 && cb >= 0 && da >= 0 && db <= 0) {
40     ans2 = 1;
41 }
42 if(ca >= 0 && cb <= 0 && da <= 0 && db >= 0) {
43     ans2 = 1;
44 }
45 return ans1 && ans2;
46 }
```

Dinic

```
1 // Network Flow
2 // Complexity  $O(n^3 \log)$  in worst case if binary search
3 // Still do pretty well without binary search
4 // Tested with: https://cses.fi/problemset/task/1695/
5 #include <bits/stdc++.h>
6
7 using namespace std;
8
9 struct Dinic {
10     struct Edge {
11         int v, pos, flow, cap;
12
13         Edge(){}
14         Edge(int _v, int _pos, int _flow, int _cap) {
15             v = _v;
16             pos = _pos;
17             flow = _flow;
18             cap = _cap;
19         }
20     };
21
22     int n;
23     vector <vector<Edge>> adj;
24     vector <int> ptr, dis;
25
26     Dinic(int _n) {
27         n = _n;
28         adj.resize(n + 5);
29         ptr = dis = vector <int> (n + 5, 0);
30     }
31
32     void addEdge(int u, int v, int cap, int rcap = 0) {
33         Edge a = Edge(v, adj[v].size(), 0, cap);
34         Edge b = Edge(u, adj[u].size(), 0, rcap);
35         adj[u].push_back(a);
36         adj[v].push_back(b);
37     }
38
39     bool bfs(int s, int t) {
40         ptr = dis = vector <int>(n + 5);
41
42         dis[s] = 1;
43         queue <int> q({s});
44         while (q.size()) {
45             int u = q.front(); q.pop();
46
47             for (Edge e : adj[u]) if (dis[e.v] == 0 && e.flow < e.cap) {
48                 dis[e.v] = dis[u] + 1;
49                 q.push(e.v);
50             }
51         }
52         return dis[t] > 0;
53     }
54 }
```

```

55 int dfs(int u, int t, int flow) {
56     if (u == t)
57         return flow;
58
59     for (int &i = ptr[u]; i < (int) adj[u].size(); i++) {
60         Edge &e = adj[u][i];
61         int dif = e.cap - e.flow;
62         if (dis[u] + 1 != dis[e.v] || dif == 0)
63             continue;
64
65         if (int val = dfs(e.v, t, min(flow, dif)) > 0) {
66             e.flow += val; adj[e.v][e.pos].flow -= val;
67             return val;
68         }
69     }
70     return 0;
71 }
72
73 void mincut(int s, int t) {
74     int res = 0;
75     while (bfs(s, t))
76         while (int val = dfs(s, t, (int) 1e9))
77             res += val;
78     cout << res << "\n";
79     bfs(s, t);
80     for (int i = 1; i <= n; i++)
81         for (Edge e : adj[i])
82             if (dis[i] > 0 && dis[e.v] == 0)
83                 cout << i << " " << e.v << "\n";
84 }
85 };

```


Max Matching

```
1 // Matching on Bipartite Graph
2 // Complexity: O(n sqrt n)
3 // Tested with: https://judge.yosupo.jp/problem/bipartitematching
4 // Note: to achieve "arbitrary" (not maximum) flow with lower bound
5 // :
6 // - Create a new graph with a new source s' and a new sink t'
7 // - if there is an edge from u => v with cap(u, v) >= dem(u, v)
8 // - cap(s', v) += dem(u, v)
9 // - cap(u, t') += dem(u, v)
10 // - cap(t, s) += oo
11 // - cap(u, v) += cap(u, v) - dem(u, v)
12 #include <bits/stdc++.h>
13 using namespace std;
14 const int N = 1e5 + 5;
15 const int oo = 1e9;
16
17 struct Hopcroft {
18     int n,m,t;
19     vector <vector <int>> adj;
20     vector <int> vis, dis, match, matched;
21
22     Hopcroft(int _n, int _m) {
23         n = _n;
24         m = _m;
25         adj.resize(n + 5);
26         vis = dis = matched = vector <int> (n + 5, 0);
27         match.assign(m + 5, -1);
28     }
29
30     void add(int u, int v) {
31         adj[u].push_back(v);
32     }
33
34     bool bfs() {
35         for (int i = 0; i < n; i++)
36             dis[i] = oo;
37
38         queue <int> q;
39
40         for (int i = 0; i < n; i++)
41             if (!matched[i])
42                 q.push(i), dis[i] = 0;
43
44         bool ok = false;
45         while (q.size()) {
46             int u = q.front(); q.pop();
47
48             for (int v : adj[u]) {
49                 if (match[v] < 0)
50                     ok = true;
51                 else if (dis[match[v]] == oo) {
52                     dis[match[v]] = dis[u] + 1;
53                     q.push(match[v]);
54                 }
55             }
56         }
57         return ok;
58     }
59
60     int max_flow() {
61         int flow = 0;
62         while (bfs()) {
63             for (int i = 0; i < n; i++)
64                 if (!matched[i])
65                     flow++;
66         }
67         return flow;
68     }
69 }
```

```

54     }
55   }
56 }
57
58   return ok;
59 }
60
61 bool dfs(int u) {
62   if (vis[u] == t)
63     return false;
64   vis[u] = t;
65   for (int v : adj[u]) if (match[v] == -1) {
66     match[v] = u;
67     matched[u] = true;
68     return true;
69   }
70
71   for (int v : adj[u]) if (match[v] != -1 && dis[match[v]] == dis
72   [u] + 1 && dfs(match[v])) {
73     match[v] = u;
74     matched[u] = true;
75     return true;
76   }
77   return false;
78 }
79
80 void maxmatch() {
81   int res = 0;
82   t = 0;
83   while (bfs()) {
84     t++;
85     for (int i = 0; i < n; i++) {
86       if (!matched[i] && dfs(i))
87         res++;
88     }
89   }
90   // cout << res << "\n";
91
92   // for (int i = 0; i < m; i++)
93   //   if (match[i] >= 0)
94   //     cout << match[i] << " " << i << "\n";
95 }
};

```

MCMF

```
1 // Min cost Max Flow
2 // Tested with: https://codeforces.com/contest/237/problem/E
3
4 #include <bits/stdc++.h>
5 #define fi first
6 #define se second
7 #define mp make_pair
8 using namespace std;
9 template <typename T1, typename T2> bool mini(T1 &a, T2 b) {
10     if (a > b) {a = b; return true;} return false;
11 }
12 template <typename T1, typename T2> bool maxi(T1 &a, T2 b) {
13     if (a < b) {a = b; return true;} return false;
14 }
15 const int N = 1e5 + 5;
16 const int oo = 1e9;
17
18 struct MCMF {
19     struct edge{
20         int u,v, flow, cap, cost;
21         edge(int _u, int _v, int _cap, int _cost) {
22             u = _u;
23             v = _v;
24             cap = _cap;
25             cost = _cost;
26             flow = 0;
27         }
28
29         int rem() {
30             return cap - flow;
31         }
32     };
33
34     int n;
35     vector <bool> inq;
36     vector <int> ptr;
37     vector <int> dis;
38     vector <vector <int>> adj;
39     vector <edge> g;
40
41     void init(int _n) {
42         n = _n;
43         ptr.resize(n + 5);
44         adj.resize(n + 5);
45         inq.resize(n + 5);
46         dis.resize(n + 5);
47     }
48
49     void addEdge(int u, int v, int cap, int cost) {
50         adj[u].push_back(g.size());
51         g.push_back(edge(u, v, cap, cost));
52         adj[v].push_back(g.size());
53         g.push_back(edge(v, u, 0, -cost));
54     }
```

```

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

```
1 // FenwickTree 2D
2 // Complexity: O(n log^2 n)
3
4 #include <bits/stdc++.h>
5 #define mp make_pair
6 #define fi first
7 #define se second
8 #define pb push_back
9 #define bit(n) (1LL << (n))
10 #define getbit(x, i) (((x) >> (i)) & 1)
11 #define pii pair<int, int>
12 #define ALL(x) x.begin(), x.end()
13 using namespace std;
14 const int M = 5e5 + 5;
15 const int N = 2e5 + 5;
16 const int mod = 1e9 + 7;
17 const int oo = 1e9;
18 const long long ooo = 1e18;
19
20 const double pi = acos(-1);
21 template<typename T1, typename T2> bool mini(T1 &a, T2 b) {if(a > b
22 ) a = b; else return 0; return 1;}
23 template<typename T1, typename T2> bool maxi(T1 &a, T2 b) {if(a < b
24 ) a = b; else return 0; return 1;}
25
26 typedef long long ll;
27
28 struct BIT2D {
29     int n;
30     vector <vector <int>> mx, yval;
31
32     BIT2D(int n) {
33         this->n = n;
34         mx.resize(n + 1);
35         yval.resize(n + 1);
36     }
37
38     void fakeupdate(int x, int y) {
39         for (; x <= n; x += x & -x)
40             yval[x].push_back(y);
41     }
42
43     void init() {
44         for (int i = 1; i <= n; i++) {
45             mx[i].assign(yval[i].size() + 1, -oo);
46             sort(ALL(yval[i]));
47         }
48     }
49
50     void update(int x, int y, int val) {
51         for (; x <= n; x += x & -x) {
52             int i = upper_bound(ALL(yval[x]), y) - yval[x].begin();
53             for (; i < (int) mx[x].size(); i += i & -i) {
54                 maxi(mx[x][i], val);
55             }
56         }
57     }
58 }
```

```

53     }
54 }
55 }
56
57 int getmax(int x, int y) {
58     int res = -oo;
59     for (; x; x -= x & -x) {
60         int i = upper_bound(ALL(yval[x]), y) - yval[x].begin();
61         for (; i; i -= i & -i) {
62             maxi(res, mx[x][i]);
63         }
64     }
65     return res;
66 }
67 };

```

Dynamic CHT

```
1 #include <bits/stdc++.h>
2 #define ll long long
3 // source: https://github.com/niklasb/contest-algos/blob/master/
4 // convex_hull/dynamic.cpp
5 // Used in problem CS Squared Ends
6 // Problem: A is an array of n integers. The cost of subarray A[l
7 // ...r] is (A[l]-A[r])^2. Partition
8 // the array into K subarrays having a minimum total cost
9 // In case of initializing 'ans', check if 1e18 is enough. Might
10 // need LLONG_MAX
11
12 using namespace std;
13 const ll is_query = -(1LL<<62);
14 struct Line {
15     ll m, b;
16     mutable function<const Line*> succ;
17     bool operator<(const Line& rhs) const {
18         if (rhs.b != is_query) return m < rhs.m;
19         const Line* s = succ();
20         if (!s) return 0;
21         ll x = rhs.m;
22         return b - s->b < (s->m - m) * x;
23     }
24 };
25 struct HullDynamic : public multiset<Line> { // will maintain upper
26     hull for maximum
27     bool bad(iterator y) {
28         auto z = next(y);
29         if (y == begin()) {
30             if (z == end()) return 0;
31             return y->m == z->m && y->b <= z->b;
32         }
33         auto x = prev(y);
34         if (z == end()) return y->m == x->m && y->b <= x->b;
35
36         // **** May need long double typecasting here
37         return (long double)(x->b - y->b)*(z->m - y->m) >= (long
38 double)(y->b - z->b)*(y->m - x->m);
39     }
40     void insert_line(ll m, ll b) {
41         auto y = insert({ m, b });
42         y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
43         if (bad(y)) { erase(y); return; }
44         while (next(y) != end() && bad(next(y))) erase(next(y));
45         while (y != begin() && bad(prev(y))) erase(prev(y));
46     }
47     ll eval(ll x) {
48         auto l = *lower_bound((Line) { x, is_query });
49         return l.m * x + l.b;
50     }
51 };
```

Persistent Dynamic Segment Tree

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 struct node {
4     int val = 0;
5     node *l = nullptr, *r = nullptr;
6     node() {
7
8     }
9     node(node *le, node *re) {
10         l = le, r = re;
11         if(l)
12             val += l->val;
13         if(r)
14             val += r->val;
15     }
16 };
17 struct segment_tree {
18     vector<node*> a = {new node()};
19     node *empty = new node();
20     int size = 1e9 + 1e5;
21     void update(node *nd, node *prev, int cl, int cr, int idx, int
22 val) {
23         int mid = (cl + cr) / 2;
24         if(cl == cr) {
25             nd->val = prev->val + val;
26         }
27         else if(idx <= mid) {
28             nd->l = new node();
29             nd->r = prev->r;
30             if(prev->l != nullptr)
31                 update(nd->l, prev->l, cl, mid, idx, val);
32             else
33                 update(nd->l, empty, cl, mid, idx, val);
34             if(nd->l)
35                 nd->val += nd->l->val;
36             if(nd->r)
37                 nd->val += nd->r->val;
38         }
39         else {
40             nd->l = prev->l;
41             nd->r = new node();
42             if(prev->r != nullptr)
43                 update(nd->r, prev->r, mid + 1, cr, idx, val);
44             else
45                 update(nd->r, empty, mid + 1, cr, idx, val);
46             if(nd->l)
47                 nd->val += nd->l->val;
48             if(nd->r)
49                 nd->val += nd->r->val;
50         }
51     }
52     void update(int idx, int val) {
53         a.push_back(new node());
54         update(a[a.size() - 1], a[a.size() - 2], 1, size, idx, val)
```



```

54     };
55 }
56 int query(node *l, node *r, int cl, int cr, int k) {
57     int mid = (cl + cr) / 2;
58     int tmpr = 0, tmp1 = 0;
59     if(cl == cr)
60         return cl;
61     if(r->l != nullptr)
62         tmpr = r->l->val;
63     if(l->l != nullptr)
64         tmp1 = l->l->val;
65     // count di kiri ga sampe k
66     // berarti k nya di kanan
67     if(tmpr - tmp1 < k) {
68         if(l->r != nullptr)
69             return query(l->r, r->r, mid + 1, cr, k - (tmpr -
70 tmp1));
71         else
72             return query(empty, r->r, mid + 1, cr, k - (tmpr -
73 tmp1));
74     }
75     else {
76         if(l->l != nullptr)
77             return query(l->l, r->l, cl, mid, k);
78         else
79             return query(empty, r->l, cl, mid, k);
80     }
81 }
82 int query(int l, int r, int k) {
83     return query(a[l - 1], a[r], 1, size, k);
84 }
85 };

```

String Hash

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 struct string_hash {
4     int lim, mod_sz, key;
5     long long power[mod_sz][lim];
6     vector<int> val[mod_sz];
7     vector<int> mod;
8     string str;
9     void build(int Lim, int Mod_sz, int Key, vector<int> Mod) {
10         lim = Lim, mod_sz = Mod_sz, key = Key, mod = Mod;
11         long long tmp[mod_sz];
12         memset(tmp, 0, sizeof(tmp));
13         for(int i = 0; i < str.size(); ++i) {
14             for(int j = 0; j < mod_sz; ++j) {
15                 tmp[j] = tmp[j] * key + str[i];
16                 tmp[j] %= mod[j];
17                 val[j].push_back(tmp[j]);
18             }
19         }
20     }
21     vector<int> q(int l, int r) {
22         vector<int> ret;
23         if(l == 0) {
24             for(int i = 0; i < mod_sz; ++i)
25                 ret.push_back(val[i][r]);
26         }
27         else {
28             for(int i = 0; i < mod_sz; ++i) {
29                 ret.push_back(val[i][r] - ((1ll * val[i][l - 1] *
30 power[i][r - l + 1]) % mod[i]));
31                 if(ret.back() < 0)
32                     ret.back() += mod[i];
33             }
34             return ret;
35         }
36     };
```

Treap

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 struct string_hash {
4     int lim, mod_sz, key;
5     long long power[mod_sz][lim];
6     vector<int> val[mod_sz];
7     vector<int> mod;
8     string str;
9     void build(int Lim, int Mod_sz, int Key, vector<int> Mod) {
10         lim = Lim, mod_sz = Mod_sz, key = Key, mod = Mod;
11         long long tmp[mod_sz];
12         memset(tmp, 0, sizeof(tmp));
13         for(int i = 0; i < str.size(); ++i) {
14             for(int j = 0; j < mod_sz; ++j) {
15                 tmp[j] = tmp[j] * key + str[i];
16                 tmp[j] %= mod[j];
17                 val[j].push_back(tmp[j]);
18             }
19         }
20     }
21     vector<int> q(int l, int r) {
22         vector<int> ret;
23         if(l == 0) {
24             for(int i = 0; i < mod_sz; ++i)
25                 ret.push_back(val[i][r]);
26         }
27         else {
28             for(int i = 0; i < mod_sz; ++i) {
29                 ret.push_back(val[i][r] - ((1ll * val[i][l - 1] *
30 power[i][r - l + 1]) % mod[i]));
31                 if(ret.back() < 0)
32                     ret.back() += mod[i];
33             }
34             return ret;
35         }
36 };

```

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

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 typedef tree<int, null_type, less_equal<int>, rb_tree_tag,
5             tree_order_statistics_node_update> ordered_multiset;
6 typedef tree<int, null_type, less<int>, rb_tree_tag,
7             tree_order_statistics_node_update> ordered_set;
8 ordered_set s;
9 s.insert(10);
10 s.insert(30);
11 s.insert(40);
12 s.insert(50);
13 s.insert(35);
14 s.order_of_key(30); // 1
15 s.order_of_key(36); // 3
16 *s.find_by_order(2); // 35
17 *s.find_by_order(0); // 10
18 const int RANDOM = chrono::high_resolution_clock::now().
19             time_since_epoch().count();
20 struct chash {
21     int operator()(int x) const { return x ^ RANDOM; }
22 };
23 gp_hash_table<key, int, chash> table;
```

Matrix Mod Expo

```
1 template<typename T> struct Matrix {
2     int sz;
3     vector<vector<T>> val;
4
5     Matrix(int _sz, bool isIdentity = false) {
6         sz = _sz;
7         val.assign(_sz, vector<T>(_sz, 0));
8         if (isIdentity) {
9             for (int i = 0; i < _sz; i++) {
10                 val[i][i] = 1;
11             }
12         }
13     }
14
15     Matrix operator * (const Matrix &other) const {
16         Matrix ret = Matrix(sz);
17         for (int i = 0; i < sz; i++) {
18             for (int j = 0; j < sz; j++) {
19                 for (int k = 0; k < sz; k++) {
20                     maddto<T>(ret.val[i][j], mmul<T>(val[i][k], other.val[k][
21                     j], mod), mod);
22                 }
23             }
24             return ret;
25         }
26
27         void print() {
28             for (int i = 0; i < sz; i++) {
29                 for (int j = 0; j < sz; j++) {
30                     cout << val[i][j] << " ";
31                 }
32                 cout << '\n';
33             }
34         }
35     };
36
37     template<typename T> Matrix<T> matexpo(Matrix<T> x, ll y) {
38         if (!y) {return Matrix<T>(x.sz, 1);}
39
40         Matrix<T> t = matexpo(x, y >> 1);
41         t = t * t;
42         if (y & 1ll) {t = t * x;}
43         return t;
44     }
```

Operator Overloading

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct Edge {
5     int a, b, w;
6     bool operator<(const Edge &y) const { return w < y.w; }
7 };
8
9 int main() {
10     int M = 4;
11     set<Edge> v;
12     for (int i = 0; i < M; ++i) {
13         int a, b, w;
14         cin >> a >> b >> w;
15         v.insert({a, b, w});
16     }
17     for (Edge e : v) cout << e.a << " " << e.b << " " << e.w << "\n";
18 }
```