

Competitive Programming Notebook

As Meninas Superpoderosas

Contents

1 Graph	2	5.16 First True	14
1.1 Bfs	2	5.17 Xor Basis	14
1.2 Floyd Warshall	2	6 String	14
1.3 Min Cost Max Flow	2	6.1 Split	14
1.4 2sat	3	6.2 Hash	14
1.5 Lca	4	6.3 Is Substring	15
1.6 Dinic	5	6.4 Trie Xor	15
1.7 Dijkstra	6	7 DP	16
1.8 3sat	6	7.1 Digit Dp	16
1.9 Ford Fulkerson	7	7.2 Lcs	16
1.10 Has Negative Cycle	7	7.3 Lis Binary Search	16
2 Primitives	8	7.4 Knapsack	17
2.1 Set Union Intersection	8	7.5 Edit Distance	17
3 Geometry	8	7.6 Digit Dp 2	18
3.1 Convex Hull	8	7.7 Lis Segtree	18
4 Math	9	7.8 Range Dp	18
4.1 Division Trick	9	8 DS	19
4.2 Log Any Base	9	8.1 Range Color Update	19
4.3 Fft Quirino	9	8.2 Trie Old	19
4.4 Factorization	9	8.3 Treap	19
4.5 Ifac	9	8.4 Sparse	20
4.6 Sieve	9	8.5 Mex	20
4.7 Ceil	10	8.6 Bit	20
4.8 Fexp	10	8.7 Maxqueue	21
4.9 Is Prime	10	8.8 Dsu	21
4.10 Divisors	10	8.9 Segtree	21
4.11 Number Sum Product Of Divisors	10	8.10 Seglazystuctnode	22
5 General	11	8.11 Mergesorttree	24
5.1 Kosaraju	11	8.12 Seghash	24
5.2 Min Priority Queue	11	8.13 Segtree Lazy Iterative	25
5.3 Random	11	8.14 Seglazy	26
5.4 Next Permutation	11	8.15 Bit2d	26
5.5 Base Converter	12	8.16 Ordered Set	27
5.6 Interactive	12	8.17 Cht	27
5.7 Flags	12	8.18 Bigk	27
5.8 Get Subsets Sum Iterative	12	8.19 Querytree	28
5.9 Last True	12	8.20 Manhattan Mst	29
5.10 Xor 1 To N	13	8.21 Trie	30
5.11 Custom Unordered Map	13	8.22 Triexor	30
5.12 Input By File	13	8.23 Kruskal	31
5.13 Mix Hash	13	8.24 Psum2d	31
5.14 Template	13		
5.15 Overflow	13		

1 Graph

1.1 Bfs

```

1 vector<vector<int>> adj; // adjacency list
  representation
2 int n; // number of nodes
3 int s; // source vertex
4
5 queue<int> q;
6 vector<bool> used(n + 1);
7 vector<int> d(n + 1), p(n + 1);
8
9 q.push(s);
10 used[s] = true;
11 p[s] = -1;
12 while (!q.empty()) {
13     int v = q.front();
14     q.pop();
15     for (int u : adj[v]) {
16         if (!used[u]) {
17             used[u] = true;
18             q.push(u);
19             d[u] = d[v] + 1;
20             p[u] = v;
21         }
22     }
23 }
24
25 // restore path
26 if (!used[u]) {
27     cout << "No path!";
28 } else {
29     vector<int> path;
30
31     for (int v = u; v != -1; v = p[v])
32         path.push_back(v);
33
34     reverse(path.begin(), path.end());
35
36     cout << "Path: ";
37     for (int v : path)
38         cout << v << " ";
39 }

```

1.2 Floyd Warshall

```

1 const long long LLINF = 0x3f3f3f3f3f3f3f3fLL;
2
3 for (int i = 0; i < n; i++) {
4     for (int j = 0; j < n; j++) {
5         adj[i][j] = 0;
6     }
7 }
8
9 long long dist[MAX][MAX];
10 for (int i = 0; i < n; i++) {
11     for (int j = 0; j < n; j++) {
12         if (i == j)
13             dist[i][j] = 0;
14         else if (adj[i][j])
15             dist[i][j] = adj[i][j];
16         else
17             dist[i][j] = LLINF;
18     }
19 }
20
21 for (int k = 0; k < n; k++) {
22     for (int i = 0; i < n; i++) {
23         for (int j = 0; j < n; j++) {
24             dist[i][j] = min(dist[i][j], dist[i][k] +
25                             dist[k][j]);
26         }
27     }
28 }

```

```

25     }
26 }
27 }

```

1.3 Min Cost Max Flow

```

1 // Min Cost Max Flow (brunomaletta)
2 //
3 // min_cost_flow(s, t, f) computa o par (fluxo, custo)
4 // com max(flujo) <= f que tenha min(custo)
5 // min_cost_flow(s, t) -> Fluxo maximo de custo
  minimo de s pra t
6 // Se for um dag, da pra substituir o SPFA por uma DP
  pra nao
7 // pagar O(nm) no comeco
8 // Se nao tiver aresta com custo negativo, nao
  precisa do SPFA
9 //
10 // O(nm + f * m log n)
11
12 template<typename T> struct mcmf {
13     struct edge {
14         int to, rev, flow, cap; // para, id da
15         reversa, fluxo, capacidade
16         bool res; // se eh reversa
17         T cost; // custo da unidade de fluxo
18         edge() : to(0), rev(0), flow(0), cap(0), cost
19         (0), res(false) {}
20         edge(int to_, int rev_, int flow_, int cap_,
21             T cost_, bool res_)
22             : to(to_), rev(rev_), flow(flow_), cap(
23             cap_), res(res_), cost(cost_) {}
24     };
25
26     vector<vector<edge>> g;
27     vector<int> par_idx, par;
28     T inf;
29     vector<T> dist;
30
31     mcmf(int n) : g(n), par_idx(n), par(n), inf(
32     numeric_limits<T>::max()/3) {}
33
34     void add(int u, int v, int w, T cost) { // de u
35     pra v com cap w e custo cost
36         edge a = edge(v, g[v].size(), 0, w, cost,
37         false);
38         edge b = edge(u, g[u].size(), 0, 0, -cost,
39         true);
40
41         g[u].push_back(a);
42         g[v].push_back(b);
43     }
44
45     vector<T> spfa(int s) { // nao precisa se nao
46     tiver custo negativo
47         deque<int> q;
48         vector<bool> is_inside(g.size(), 0);
49         dist = vector<T>(g.size(), inf);
50
51         dist[s] = 0;
52         q.push_back(s);
53         is_inside[s] = true;
54
55         while (!q.empty()) {
56             int v = q.front();
57             q.pop_front();
58             is_inside[v] = false;
59
60             for (int i = 0; i < g[v].size(); i++) {
61                 auto [to, rev, flow, cap, res, cost]
62                 = g[v][i];

```

```

53         if (flow < cap and dist[v] + cost <
54             dist[to]) {
55             dist[to] = dist[v] + cost;
56             if (is_inside[to]) continue;
57             if (!q.empty() and dist[to] >
58                 dist[q.front()]) q.push_back(to);
59             else q.push_front(to);
60             is_inside[to] = true;
61         }
62     }
63     return dist;
64 }
65 bool dijkstra(int s, int t, vector<T>& pot) {
66     priority_queue<pair<T, int>, vector<pair<T,
67         int>>, greater<>> q;
68     dist = vector<T>(g.size(), inf);
69     dist[s] = 0;
70     q.emplace(0, s);
71     while (q.size()) {
72         auto [d, v] = q.top();
73         q.pop();
74         if (dist[v] < d) continue;
75         for (int i = 0; i < g[v].size(); i++) {
76             auto [to, rev, flow, cap, res, cost]
77                 = g[v][i];
78             cost += pot[v] - pot[to];
79             if (flow < cap and dist[v] + cost <
80                 dist[to]) {
81                 dist[to] = dist[v] + cost;
82                 q.emplace(dist[to], to);
83                 par_idx[to] = i, par[to] = v;
84             }
85         }
86     }
87     return dist[t] < inf;
88 }
89
90 pair<int, T> min_cost_flow(int s, int t, int flow
91     = INF) {
92     vector<T> pot(g.size(), 0);
93     pot = spfa(s); // mudar algoritmo de caminho
94     minimo aqui
95
96     int f = 0;
97     T ret = 0;
98     while (f < flow and dijkstra(s, t, pot)) {
99         for (int i = 0; i < g.size(); i++)
100             if (dist[i] < inf) pot[i] += dist[i];
101
102         int mn_flow = flow - f, u = t;
103         while (u != s){
104             mn_flow = min(mn_flow,
105                 g[par[u]][par_idx[u]].cap - g[pa
106 [u]][par_idx[u]].flow);
107             u = par[u];
108         }
109
110         ret += pot[t] * mn_flow;
111
112         u = t;
113         while (u != s) {
114             g[par[u]][par_idx[u]].flow += mn_flow;
115             g[u][g[par[u]][par_idx[u]].rev].flow
116 -= mn_flow;
117             u = par[u];
118         }
119
120         f += mn_flow;
121     }
122 }

```

```

116         return make_pair(f, ret);
117     }
118
119     // Opcional: retorna as arestas originais por
120     // onde passa flow = cap
121     vector<pair<int,int>> recover() {
122         vector<pair<int,int>> used;
123         for (int i = 0; i < g.size(); i++) for (edge
124             e : g[i])
125             if(e.flow == e.cap && !e.res) used.
126             push_back({i, e.to});
127         return used;
128     }
129 };

```

1.4 2sat

```

1 // 2SAT
2 //
3 // verifica se existe e encontra soluÃ§Ã£o
4 // para fÃ³rmulas booleanas da forma
5 // (a or b) and (!a or c) and (...)
6 //
7 // indexado em 0
8 // n(a) = 2*x e n(~a) = 2*x+1
9 // a = 2 ; n(a) = 4 ; n(~a) = 5 ; n(a)^1 = 5 ; n(~a)
  ^1 = 4
10 //
11 // https://cses.fi/problemset/task/1684/
12 // https://codeforces.com/gym/104120/problem/E
13 // (add_eq, add_true, add_false e at_most_one nÃ£o
  foram testadas)
14 //
15 // O(n + m)
16
17 struct sat {
18     int n, tot;
19     vector<vector<int>> adj, adjt; // grafo original,
      grafo transposto
20     vector<int> vis, comp, ans;
21     stack<int> topo; // ordem topolÃ³gica
22
23     sat() {}
24     sat(int n_) : n(n_), tot(n), adj(2*n), adjt(2*n)
      {}
25
26     void dfs(int x) {
27         vis[x] = true;
28
29         for (auto e : adj[x]) {
30             if (!vis[e]) dfs(e);
31         }
32
33         topo.push(x);
34     }
35
36     void dfst(int x, int& id) {
37         vis[x] = true;
38         comp[x] = id;
39
40         for (auto e : adjt[x]) {
41             if (!vis[e]) dfst(e, id);
42         }
43     }
44
45     void add_impl(int a, int b) { // a -> b = (!a or
      b)
46         a = (a >= 0 ? 2*a : -2*a-1);
47         b = (b >= 0 ? 2*b : -2*b-1);
48
49         adj[a].push_back(b);
50         adj[b^1].push_back(a^1);
51     }

```

```

52     adjt[b].push_back(a);
53     adjt[a^1].push_back(b^1);
54 }
55
56 void add_or(int a, int b) { // a or b
57     add_impl(~a, b);
58 }
59
60 void add_nor(int a, int b) { // a nor b = !(a or
61     b)
62     add_or(~a, b), add_or(a, ~b), add_or(~a, ~b);
63 }
64
65 void add_and(int a, int b) { // a and b
66     add_or(a, b), add_or(~a, b), add_or(a, ~b);
67 }
68
69 void add_nand(int a, int b) { // a nand b = !(a
70     and b)
71     add_or(~a, ~b);
72 }
73
74 void add_xor(int a, int b) { // a xor b = (a != b)
75     add_or(a, b), add_or(~a, ~b);
76 }
77
78 void add_xnor(int a, int b) { // a xnor b = !(a
79     xor b) = (a == b)
80     add_xor(~a, b);
81 }
82
83 void add_true(int a) { // a = T
84     add_or(a, ~a);
85 }
86
87 void add_false(int a) { // a = F
88     add_and(a, ~a);
89 }
90
91 // magia - brunomaleto
92 void add_true_old(int a) { // a = T (n sei se
93     funciona)
94     add_impl(~a, a);
95 }
96
97 void at_most_one(vector<int> v) { // no max um
98     verdadeiro
99     adj.resize(2*(tot+v.size()));
100     for (int i = 0; i < v.size(); i++) {
101         add_impl(tot+i, ~v[i]);
102         if (i) {
103             add_impl(tot+i, tot+i-1);
104             add_impl(v[i], tot+i-1);
105         }
106     }
107     tot += v.size();
108 }
109
110 pair<bool, vector<int>> solve() {
111     ans.assign(n, -1);
112     comp.assign(2*tot, -1);
113     vis.assign(2*tot, 0);
114     int id = 1;
115     for (int i = 0; i < 2*tot; i++) if (!vis[i])
116         dfs(i);
117
118     vis.assign(2*tot, 0);
119     while (topo.size()) {
120         auto x = topo.top();
121         topo.pop();

```

```

118     if (!vis[x]) {
119         dfst(x, id);
120         id++;
121     }
122 }
123
124 for (int i = 0; i < tot; i++) {
125     if (comp[2*i] == comp[2*i+1]) return {
126         false, {} };
127     ans[i] = (comp[2*i] > comp[2*i+1]);
128 }
129
130 return {true, ans};
131 }
132 };

```

1.5 Lca

```

1 // LCA
2 //
3 // lowest common ancestor between two nodes
4 //
5 // edit_distance(n, adj, root)
6 //
7 // https://cses.fi/problemset/task/1688
8 //
9 // O(log N)
10
11 struct LCA {
12     const int MAXE = 31;
13     vector<vector<int>>> up;
14     vector<int> dep;
15
16     LCA(int n, vector<vector<int>>& adj, int root =
17         1) {
18         up.assign(n+1, vector<int>(MAXE, -1));
19         dep.assign(n+1, 0);
20
21         dep[root] = 1;
22         dfs(root, -1, adj);
23
24         for (int j = 1; j < MAXE; j++) {
25             for (int i = 1; i <= n; i++) {
26                 if (up[i][j-1] != -1)
27                     up[i][j] = up[ up[i][j-1] ][j-1];
28             }
29         }
30     }
31
32     void dfs(int x, int p, vector<vector<int>>& adj)
33     {
34         up[x][0] = p;
35         for (auto e : adj[x]) {
36             if (e != p) {
37                 dep[e] = dep[x] + 1;
38                 dfs(e, x, adj);
39             }
40         }
41     }
42
43     int jump(int x, int k) { // jump from node x k
44         times
45         for (int i = 0; i < MAXE; i++) {
46             if (k & (1 << i) && x != -1) x = up[x][i];
47         }
48         return x;
49     }
50
51     int lca(int a, int b) {
52         if (dep[a] > dep[b]) swap(a, b);
53         b = jump(b, dep[b] - dep[a]);
54
55         if (a == b) return a;

```

```

53         for (int i = MAXE-1; i >= 0; i--) {
54             if (up[a][i] != up[b][i]) {
55                 a = up[a][i];
56                 b = up[b][i];
57             }
58         }
59     }
60
61     return up[a][0];
62 }
63
64 int dist(int a, int b) {
65     return dep[a] + dep[b] - 2 * dep[lca(a, b)];
66 }
67 };

```

1.6 Dinic

```

1  // Dinic / Dinitz
2  //
3  // max-flow / min-cut
4  //
5  // https://cses.fi/problemset/task/1694/
6  //
7  // O(E * V^2)
8
9  using ll = long long;
10 const ll FLOW_INF = 1e18 + 7;
11
12 struct Edge {
13     int from, to;
14     ll cap, flow;
15     Edge* residual; // a inversa da minha aresta
16
17     Edge() {};
18
19     Edge(int from, int to, ll cap) : from(from), to(to), cap(cap), flow(0) {};
20
21     ll remaining_cap() {
22         return cap - flow;
23     }
24
25     void augment(ll bottle_neck) {
26         flow += bottle_neck;
27         residual->flow -= bottle_neck;
28     }
29
30     bool is_residual() {
31         return cap == 0;
32     }
33 };
34
35 struct Dinic {
36     int n;
37     vector<vector<Edge*>> adj;
38     vector<int> level, next;
39
40     Dinic(int n): n(n) {
41         adj.assign(n+1, vector<Edge*>());
42         level.assign(n+1, -1);
43         next.assign(n+1, 0);
44     }
45
46     void add_edge(int from, int to, ll cap) {
47         auto e1 = new Edge(from, to, cap);
48         auto e2 = new Edge(to, from, 0);
49
50         e1->residual = e2;
51         e2->residual = e1;
52
53         adj[from].push_back(e1);
54         adj[to].push_back(e2);

```

```

55     }
56
57     bool bfs(int s, int t) {
58         fill(level.begin(), level.end(), -1);
59         queue<int> q;
60
61         q.push(s);
62         level[s] = 1;
63
64         while (q.size()) {
65             int curr = q.front();
66             q.pop();
67
68             for (auto edge : adj[curr]) {
69                 if (edge->remaining_cap() > 0 &&
70                     level[edge->to] == -1) {
71                     level[edge->to] = level[curr] +
72                     1;
73                     q.push(edge->to);
74                 }
75             }
76
77             return level[t] != -1;
78         }
79
80         ll dfs(int x, int t, ll flow) {
81             if (x == t) return flow;
82
83             for (int& cid = next[x]; cid < (int)adj[x].
84                 size(); cid++) {
85                 auto& edge = adj[x][cid];
86                 ll cap = edge->remaining_cap();
87
88                 if (cap > 0 && level[edge->to] == level[x]
89                     + 1) {
90                     ll sent = dfs(edge->to, t, min(flow,
91                         cap)); // bottle neck
92                     if (sent > 0) {
93                         edge->augment(sent);
94                         return sent;
95                     }
96                 }
97             }
98
99             return 0;
100         }
101
102         ll solve(int s, int t) {
103             ll max_flow = 0;
104
105             while (bfs(s, t)) {
106                 fill(next.begin(), next.end(), 0);
107
108                 while (ll sent = dfs(s, t, FLOW_INF)) {
109                     max_flow += sent;
110                 }
111             }
112
113             return max_flow;
114         }
115
116         // path recover
117         vector<bool> vis;
118         vector<int> curr;
119
120         bool dfs2(int x, int& t) {
121             vis[x] = true;
122             bool arrived = false;
123
124             if (x == t) {
125                 curr.push_back(x);
126                 return true;

```

```

123     }
124
125     for (auto e : adj[x]) {
126         if (e->flow > 0 && !vis[e->to]) { // !e->
is_residual() &&
127             bool aux = dfs2(e->to, t);
128
129             if (aux) {
130                 arrived = true;
131                 e->flow--;
132             }
133         }
134     }
135
136     if (arrived) curr.push_back(x);
137
138     return arrived;
139 }
140
141 vector<vector<int>> get_paths(int s, int t) {
142     vector<vector<int>> ans;
143
144     while (true) {
145         curr.clear();
146         vis.assign(n+1, false);
147
148         if (!dfs2(s, t)) break;
149
150         reverse(curr.begin(), curr.end());
151         ans.push_back(curr);
152     }
153
154     return ans;
155 }
156 };

```

1.7 Dijkstra

```

1 const int INF = 1e9+17;
2 vector<vector<pair<int, int>>> adj; // {neighbor,
   weight}
3
4 void dijkstra(int s, vector<int> & d, vector<int> & p
   ) {
5     int n = adj.size();
6     d.assign(n, INF);
7     p.assign(n, -1);
8
9     d[s] = 0;
10    set<pair<int, int>> q;
11    q.insert({0, s});
12    while (!q.empty()) {
13        int v = q.begin()->second;
14        q.erase(q.begin());
15
16        for (auto edge : adj[v]) {
17            int to = edge.first;
18            int len = edge.second;
19
20            if (d[v] + len < d[to]) {
21                q.erase({d[to], to});
22                d[to] = d[v] + len;
23                p[to] = v;
24                q.insert({d[to], to});
25            }
26        }
27    }
28 }

```

1.8 3sat

```

1 // We are given a CNF, e.g.  $\phi(x) = (x_1 \text{ or } \sim x_2)$ 
   and  $(x_3 \text{ or } \sim x_4 \text{ or } \sim x_5)$  and ...

```

```

2 // SAT finds an assignment x for  $\phi(x) = \text{true}$ .
3 // Davis-Putnam-Logemann-Loveland Algorithm (
   youknowwho code)
4 // Complexity:  $O(2^n)$  in worst case.
5 // This implementation is practical for  $n \leq 1000$  or
   more. lmao.
6
7 #include<bits/stdc++.h>
8 using namespace std;
9
10 const int N = 3e5 + 9;
11
12 // positive literal x in [0,n),
13 // negative literal ~x in [-n,0)
14 // 0 indexed
15 struct SAT_GOD {
16     int n;
17     vector<int> occ, pos, neg;
18     vector<vector<int>> g, lit;
19     SAT_GOD(int n) : n(n), g(2*n), occ(2*n) { }
20     void add_clause(const vector<int> &c) {
21         for (auto u : c) {
22             g[u+n].push_back(lit.size());
23             occ[u+n] += 1;
24         }
25         lit.push_back(c);
26     }
27     //(u | v | ~w) -> (u, 0, v, 1, w, 0)
28     void add(int u, int af, int v = 1e9, int bf = 0,
        int w = 1e9, int cf = 0) {
29         vector<int> a;
30         if (!af) u = ~u;
31         a.push_back(u);
32         if (v != 1e9) {
33             if (!bf) v = ~v;
34             a.push_back(v);
35         }
36         if (w != 1e9) {
37             if (!cf) w = ~w;
38             a.push_back(w);
39         }
40         add_clause(a);
41     }
42     vector<bool> x;
43     vector<vector<int>> decision_stack;
44     vector<int> unit_stack, pure_stack;
45     void push(int u) {
46         x[u + n] = 1;
47         decision_stack.back().push_back(u);
48         for (auto i : g[u + n]) if (pos[i]++ == 0) {
49             for (auto u : lit[i]) --occ[u+n];
50         }
51         for (auto i : g[~u + n]) {
52             ++neg[i];
53             if (pos[i] == 0) unit_stack.push_back(i);
54         }
55     }
56     void pop() {
57         int u = decision_stack.back().back();
58         decision_stack.back().pop_back();
59         x[u + n] = 0;
60         for (auto i : g[u + n]) if (--pos[i] == 0) {
61             for (auto u : lit[i]) ++occ[u + n];
62         }
63         for (auto i : g[~u+n]) --neg[i];
64     }
65     bool reduction() {
66         while(!unit_stack.empty() || !pure_stack.empty())
67         {
68             if(!pure_stack.empty()) { // pure literal
69                 elimination

```

```

70         if (occ[u + n] == 1 && occ[~u + n] == 0) push
(u);
71     } else { // unit propagation
72         int i = unit_stack.back();
73         unit_stack.pop_back();
74         if(pos[i] > 0) continue;
75         if(neg[i] == lit[i].size()) return false;
76         if(neg[i] + 1 == lit[i].size()) {
77             int w = n;
78             for (int u: lit[i]) if (!x[u + n] && !x[~u
+ n]) w = u;
79             if (x[~w + n]) return false;
80             push(w);
81         }
82     }
83 }
84 return true;
85 }
86 bool ok() {
87     x.assign(2*n,0);
88     pos = neg = vector<int>(lit.size());
89     decision_stack.assign(1, {});
90     while(1) {
91         if(reduction()) {
92             int s = 0;
93             for(int u = 0; u < n; ++u) if(occ[s + n] +
occ[~s + n] < occ[u + n] + occ[~u + n]) s = u;
94             if(occ[s + n] + occ[~s + n] == 0) return true
;
95             decision_stack.push_back({});
96             push(s);
97         } else {
98             int s = decision_stack.back()[0];
99             while(!decision_stack.back().empty()) pop();
100             decision_stack.pop_back();
101             if (decision_stack.empty()) return false;
102             push(~s);
103         }
104     }
105 }
106 };
107
108 int32_t main() {
109     int n = 9;
110     SAT_GOD t(n);
111     t.add(0, 0, 1, 1);
112     t.add(1, 0);
113     t.add(1, 0, 3, 1, 5, 1);
114     cout << t.ok() << endl;
115 }

```

1.9 Ford Fulkerson

```

1 // Ford-Fulkerson
2 //
3 // max-flow / min-cut
4 //
5 // MAX nÃŕs
6 //
7 // https://cses.fi/problemset/task/1694/
8 //
9 // O(m * max_flow)
10
11 using ll = long long;
12 const int MAX = 510;
13
14 struct Flow {
15     int n;
16     ll adj[MAX][MAX];
17     bool used[MAX];
18
19     Flow(int n) : n(n) {};
20

```

```

void add_edge(int u, int v, ll c) {
    adj[u][v] += c;
    adj[v][u] = 0; // cuidado com isso
}

ll dfs(int x, int t, ll amount) {
    used[x] = true;

    if (x == t) return amount;

    for (int i = 1; i <= n; i++) {
        if (adj[x][i] > 0 && !used[i]) {
            ll sent = dfs(i, t, min(amount, adj[x
][i]));

            if (sent > 0) {
                adj[x][i] -= sent;
                adj[i][x] += sent;

                return sent;
            }
        }
    }

    return 0;
}

ll max_flow(int s, int t) { // source and sink
    ll total = 0;
    ll sent = -1;

    while (sent != 0) {
        memset(used, 0, sizeof(used));
        sent = dfs(s, t, INT_MAX);
        total += sent;
    }

    return total;
}
};

```

1.10 Has Negative Cycle

```

1 // Edson
2
3 using edge = tuple<int, int, int>;
4
5 bool has_negative_cycle(int s, int N, const vector<
edge>& edges)
6 {
7     const int INF { 1e9+17 };
8
9     vector<int> dist(N + 1, INF);
10    dist[s] = 0;
11
12    for (int i = 1; i <= N - 1; i++) {
13        for (auto [u, v, w] : edges) {
14            if (dist[u] < INF && dist[v] > dist[u] +
w) {
15                dist[v] = dist[u] + w;
16            }
17        }
18    }
19
20    for (auto [u, v, w] : edges) {
21        if (dist[u] < INF && dist[v] > dist[u] + w) {
22            return true;
23        }
24    }
25
26    return false;
27 }

```

2 Primitives

2.1 Set Union Intersection

```

1 // Template pra fazer união e intercessão de sets
  de forma fácil
2 // Usar + para uniao e * para intercessão
3 // Source: https://stackoverflow.com/questions
  /13448064/how-to-find-the-intersection-of-two-stl
  -sets
4
5 template <class T, class CMP = std::less<T>, class
  ALLOC = std::allocator<T> >
6 std::set<T, CMP, ALLOC> operator * (
7     const std::set<T, CMP, ALLOC> &s1, const std::set<T
  , CMP, ALLOC> &s2)
8 {
9     std::set<T, CMP, ALLOC> s;
10    std::set_intersection(s1.begin(), s1.end(), s2.
  begin(), s2.end(),
11        std::inserter(s, s.begin()));
12    return s;
13 }
14
15 template <class T, class CMP = std::less<T>, class
  ALLOC = std::allocator<T> >
16 std::set<T, CMP, ALLOC> operator + (
17     const std::set<T, CMP, ALLOC> &s1, const std::set<T
  , CMP, ALLOC> &s2)
18 {
19     std::set<T, CMP, ALLOC> s;
20    std::set_union(s1.begin(), s1.end(), s2.begin(), s2
  .end(),
21        std::inserter(s, s.begin()));
22    return s;
23 }

```

3 Geometry

3.1 Convex Hull

```

1 // Convex Hull - Monotone Chain
2 //
3 // Convex Hull is the subset of points that forms the
  smallest convex polygon
4 // which encloses all points in the set.
5 //
6 // https://cses.fi/problemset/task/2195/
7 // https://open.kattis.com/problems/convexhull (
  counterclockwise)
8 //
9 // O(n log(n))
10
11 typedef long long ftype;
12
13 struct Point {
14     ftype x, y;
15
16     Point() {};
17     Point(ftype x, ftype y) : x(x), y(y) {};
18
19     bool operator<(Point o) {
20         if (x == o.x) return y < o.y;
21         return x < o.x;
22     }
23
24     bool operator==(Point o) {
25         return x == o.x && y == o.y;
26     }
27 };
28

```

```

29 ftype cross(Point a, Point b, Point c) {
30     // v: a -> c
31     // w: a -> b
32
33     // v: c.x - a.x, c.y - a.y
34     // w: b.x - a.x, b.y - a.y
35
36     return (c.x - a.x) * (b.y - a.y) - (c.y - a.y) *
  (b.x - a.x);
37 }
38
39 ftype dir(Point a, Point b, Point c) {
40     // 0 -> colineares
41     // -1 -> esquerda
42     // 1 -> direita
43
44     ftype cp = cross(a, b, c);
45
46     if (cp == 0) return 0;
47     else if (cp < 0) return -1;
48     else return 1;
49 }
50
51 vector<Point> convex_hull(vector<Point> points) {
52     sort(points.begin(), points.end());
53     points.erase( unique(points.begin(), points.end()
  ), points.end()); // somente pontos distintos
54     int n = points.size();
55
56     if (n == 1) return { points[0] };
57
58     vector<Point> upper_hull = {points[0], points
  [1]};
59     for (int i = 2; i < n; i++) {
60         upper_hull.push_back(points[i]);
61
62         int sz = upper_hull.size();
63
64         while (sz >= 3 && dir(upper_hull[sz-3],
  upper_hull[sz-2], upper_hull[sz-1]) == -1) {
65             upper_hull.pop_back();
66             upper_hull.pop_back();
67             upper_hull.push_back(points[i]);
68             sz--;
69         }
70     }
71
72     vector<Point> lower_hull = {points[n-1], points[n
  -2]};
73     for (int i = n-3; i >= 0; i--) {
74         lower_hull.push_back(points[i]);
75
76         int sz = lower_hull.size();
77
78         while (sz >= 3 && dir(lower_hull[sz-3],
  lower_hull[sz-2], lower_hull[sz-1]) == -1) {
79             lower_hull.pop_back();
80             lower_hull.pop_back();
81             lower_hull.push_back(points[i]);
82             sz--;
83         }
84     }
85
86     // reverse(lower_hull.begin(), lower_hull.end());
87     // counterclockwise
88     for (int i = (int)lower_hull.size() - 2; i > 0; i
  --) {
89         upper_hull.push_back(lower_hull[i]);
90     }
91
92     return upper_hull;
93 }

```


4 Math

4.1 Division Trick

```

1 for(int l = 1, r; l <= n; l = r + 1) {
2     r = n / (n / l);
3     // n / x yields the same value for l <= x <= r
4 }
5 for(int l, r = n; r > 0; r = l - 1) {
6     int tmp = (n + r - 1) / r;
7     l = (n + tmp - 1) / tmp;
8     // (n+x-1) / x yields the same value for l <= x
      <= r
9 }

```

4.2 Log Any Base

```

1 int intlog(double base, double x) {
2     return (int)(log(x) / log(base));
3 }

```

4.3 Fft Quirino

```

1 // FFT
2 //
3 // boa em memória e ok em tempo
4 //
5 // https://codeforces.com/group/YgJmumGtHD/contest
      /528947/problem/H (maratona mineira)
6
7 using cd = complex<double>;
8 const double PI = acos(-1);
9
10 void fft(vector<cd> &A, bool invert) {
11     int N = size(A);
12
13     for (int i = 1, j = 0; i < N; i++) {
14         int bit = N >> 1;
15         for (; j & bit; bit >>= 1)
16             j ^= bit;
17         j ^= bit;
18
19         if (i < j)
20             swap(A[i], A[j]);
21     }
22
23     for (int len = 2; len <= N; len <= 1) {
24         double ang = 2 * PI / len * (invert ? -1 : 1);
25         cd wlen(cos(ang), sin(ang));
26         for (int i = 0; i < N; i += len) {
27             cd w(1);
28             for (int j = 0; j < len/2; j++) {
29                 cd u = A[i+j], v = A[i+j+len/2] * w;
30                 A[i+j] = u + v;
31                 A[i+j+len/2] = u - v;
32                 w *= wlen;
33             }
34         }
35     }
36
37     if (invert) {
38         for (auto &x : A)
39             x /= N;
40     }
41 }
42
43 vector<int> multiply(vector<int> const& A, vector<int>
      > const& B) {
44     vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
45     int N = 1;

```

```

46     while (N < size(A) + size(B))
47         N <= 1;
48     fa.resize(N);
49     fb.resize(N);
50
51     fft(fa, false);
52     fft(fb, false);
53     for (int i = 0; i < N; i++)
54         fa[i] *= fb[i];
55     fft(fa, true);
56
57     vector<int> result(N);
58     for (int i = 0; i < N; i++)
59         result[i] = round(fa[i].real());
60     return result;
61 }

```

4.4 Factorization

```

1 // nson
2
3 using ll = long long;
4
5 vector<pair<ll, int>> factorization(ll n) {
6     vector<pair<ll, int>> ans;
7
8     for (ll p = 2; p*p <= n; p++) {
9         if (n%p == 0) {
10             int expoente = 0;
11
12             while (n%p == 0) {
13                 n /= p;
14                 expoente++;
15             }
16
17             ans.push_back({p, expoente});
18         }
19     }
20
21     if (n > 1) {
22         ans.push_back({n, 1});
23     }
24
25     return ans;
26 }

```

4.5 Ifac

```

1 // inverse of factorial
2
3 mint fac[N], ifac[N];
4
5 void build_fac() {
6     fac[0] = 1;
7
8     for (int i = 1; i < N; i++) {
9         fac[i] = fac[i - 1] * i;
10    }
11
12    ifac[N - 1] = inv(fac[N - 1]);
13
14    for (int i = N - 2; i >= 0; i--) {
15        ifac[i] = ifac[i + 1] * (i + 1);
16    }
17 }

```

4.6 Sieve

```

1 vector<int> sieve(int MAXN){
2     //list of prime numbers up to MAXN
3     vector<int> primes;
4     bitset<(int)1e7> not_prime;

```

```

5     not_prime[0] = 1;
6     not_prime[1] = 1;
7     for(int i = 2; i <= MAXN; i++){
8         if(!not_prime[i]){
9             primes.push_back(i);
10            for(ll j = 1LL * i * i; j <= MAXN; j += i
11        ){
12            not_prime[(int)j] = 1;
13        }
14    }
15    return primes;
16 }

```

4.7 Ceil

```

1 using ll = long long;
2
3 // avoid overflow
4 ll division_ceil(ll a, ll b) {
5     return 1 + ((a - 1) / b); // if a != 0
6 }
7
8 int intceil(int a, int b) {
9     return (a+b-1)/b;
10 }

```

4.8 Fexp

```

1 using ll = long long;
2
3 ll fexp(ll base, ll exp, ll m) {
4     ll ans = 1;
5     base %= m;
6
7     while (exp > 0) {
8         if (exp % 2 == 1) {
9             ans = (ans * base) % m;
10        }
11
12        base = (base * base) % m;
13        exp /= 2;
14    }
15
16    return ans;
17 }

```

4.9 Is Prime

```

1 bool is_prime(ll n) {
2     if (n <= 1) return false;
3     if (n == 2) return true;
4
5     for (ll i = 2; i*i <= n; i++) {
6         if (n % i == 0)
7             return false;
8     }
9
10    return true;
11 }

```

4.10 Divisors

```

1 vector<ll> divisors(ll n) {
2     vector<ll> ans;
3
4     for (ll i = 1; i*i <= n; i++) {
5         if (n%i == 0) {
6             ll value = n/i;
7
8             ans.push_back(i);

```

```

9             if (value != i) {
10                ans.push_back(value);
11            }
12        }
13    }
14
15    return ans;
16 }

```

4.11 Number Sum Product Of Divisors

```

1 // CSES - Divisor Analysis
2 // Print the number, sum and product of the divisors.
3 // Since the input number may be large, it is given
4 // as a prime factorization.
5 // Input:
6 // The first line has an integer n: the number of
7 // parts in the prime factorization.
8 // After this, there are n lines that describe the
9 // factorization. Each line has two numbers x and k
10 // where x is a prime and k is its power.
11 // Output:
12 // Print three integers modulo 10^9+7: the number,
13 // sum and product of the divisors.
14 // Constraints:
15 // (1 <= n <= 1e5) ; (2 <= x <= 1e6) ; (1 <= k <= 1e9
16 // ) ; each x is a distinct prime
17
18 #include <bits/stdc++.h>
19 typedef long long ll;
20 using namespace std;
21 const ll MOD = 1e9 + 7;
22
23 ll expo(ll base, ll pow) {
24     ll ans = 1;
25     while (pow) {
26         if (pow & 1) ans = ans * base % MOD;
27         base = base * base % MOD;
28         pow >>= 1;
29     }
30     return ans;
31 }
32
33 ll p[100001], k[100001];
34
35 int main() {
36     cin.tie(0)->sync_with_stdio(0);
37     int n;
38     cin >> n;
39     for (int i = 0; i < n; i++) cin >> p[i] >> k[i];
40     ll div_cnt = 1, div_sum = 1, div_prod = 1,
41     div_cnt2 = 1;
42     for (int i = 0; i < n; i++) {
43         div_cnt = div_cnt * (k[i] + 1) % MOD;
44         div_sum = div_sum * (expo(p[i], k[i] + 1) -
45         1) % MOD *
46         expo(p[i] - 1, MOD - 2) % MOD;
47         div_prod = expo(div_prod, k[i] + 1) *
48         expo(expo(p[i], (k[i] * (k[i] + 1)
49         / 2)), div_cnt2) % MOD;
50         div_cnt2 = div_cnt2 * (k[i] + 1) % (MOD - 1);
51     }
52     cout << div_cnt << ' ' << div_sum << ' ' <<
53     div_prod;
54     return 0;
55 }

```

5 General

5.1 Kosaraju

```

1 // https://codeforces.com/blog/entry/125435
2 #ifdef MAXWELL_LOCAL_DEBUG
3 #include "debug/debug_template.cpp"
4 #define dbg debug
5 #else
6 #define debug(...)
7 #define dbg debug
8 #define debugArr(arr, n)
9 #endif
10
11 #include <bits/stdc++.h>
12 #define ff first
13 #define ss second
14
15 using namespace std;
16 using ll = long long;
17 using ld = long double;
18 using pii = pair<int, int>;
19 using vi = vector<int>;
20
21 using tii = tuple<int, int, int>;
22 // auto [a,b,c] = ...
23 // .insert({a,b,c})
24
25 const int oo = (int)1e9 + 5; //INF to INT
26 const ll OO = 0x3f3f3f3f3f3f3f3fLL; //INF to LL
27
28 struct Kosaraju {
29
30     int N;
31     int cntComps;
32
33     vector<vector<int>> g;
34     vector<vector<int>> gi;
35
36     stack<int> S;
37     vector<int> vis;
38     vector<int> comp;
39
40     Kosaraju(vector<vector<int>>& arr) {
41         N = (int)arr.size();
42         cntComps = 0;
43
44         g.resize(N);
45         gi.resize(N);
46         vis.resize(N);
47         comp.resize(N);
48
49         for(int i = 0; i < (int)arr.size(); i++) {
50             for(auto &v : arr[i]) {
51                 g[i].push_back(v);
52                 gi[v].push_back(i);
53             }
54         }
55
56         run();
57     }
58
59     void dfs(int u) {
60         vis[u] = 1;
61         for(auto &v : g[u]) if(!vis[v]) {
62             dfs(v);
63         }
64         S.push(u);
65     }
66
67     void scc(int u, int c) {
68         vis[u] = 1;

```

```

69         comp[u] = c;
70         for(auto &v : gi[u]) if(!vis[v]) {
71             scc(v, c);
72         }
73     }
74
75     void run() {
76         vis.assign(N, 0);
77
78         for(int i = 0; i < N; i++) if(!vis[i]) {
79             dfs(i);
80         }
81
82         vis.assign(N, 0);
83
84         while((int)S.size()) {
85             int u = S.top();
86             S.pop();
87             if(!vis[u]) {
88                 scc(u, cntComps++);
89             }
90         }
91     }
92 }
93
94 };
95
96 int main() {
97     ios::sync_with_stdio(false);
98     cin.tie(NULL);
99
100     int t = 1;
101
102     while(t--) {
103         solve();
104     }
105
106 }

```

5.2 Min Priority Queue

```

1 template<class T> using min_priority_queue =
    priority_queue<T, vector<T>, greater<T>>;

```

5.3 Random

```

1 int main() {
2     ios::sync_with_stdio(false);
3     cin.tie(NULL);
4
5     //mt19937 rng(chrono::steady_clock::now().
6     time_since_epoch().count()); //gerar int
7     mt19937_64 rng(chrono::steady_clock::now().
8     time_since_epoch().count()); //gerar ll
9
10    /*usar rng() pra gerar numeros aleatÃrios.*/
11    /*usar rng() % x pra gerar numeros em [0, x-1]*/
12    for(int i = 0; i < 10; i++){
13        cout << rng() << endl;
14    }
15    vector<ll> arr = {1,2,3,4,5,6,7,8,9};
16    /*dÃ pra usar no shuffle de vector tambÃm*/
17    shuffle(arr.begin(), arr.end(), rng);
18    for(auto &x: arr)
19        cout << x << endl;
20 }

```

5.4 Next Permutation

```

1 // output: 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2; 3,2,1;
2

```

```

3 vector<int> arr = {1, 2, 3};
4 int n = arr.size();
5
6 do {
7     for (auto e : arr) {
8         cout << e << ' ';
9     }
10    cout << '\n';
11 } while (next_permutation(arr.begin(), arr.end()));

```

5.5 Base Converter

```

1 const string digits = "0123456789
  ABCDEFGHIJKLMNOPQRSTUVWXYZ";
2
3 ll tobase10(string number, int base) {
4     map<char, int> val;
5     for (int i = 0; i < digits.size(); i++) {
6         val[digits[i]] = i;
7     }
8
9     ll ans = 0, pot = 1;
10
11    for (int i = number.size() - 1; i >= 0; i--) {
12        ans += val[number[i]] * pot;
13        pot *= base;
14    }
15
16    return ans;
17 }
18
19 string frombase10(ll number, int base) {
20     if (number == 0) return "0";
21
22     string ans = "";
23
24     while (number > 0) {
25         ans += digits[number % base];
26         number /= base;
27     }
28
29     reverse(ans.begin(), ans.end());
30
31     return ans;
32 }
33
34 // verifica se um número está na base especificada
35 bool verify_base(string num, int base) {
36     map<char, int> val;
37     for (int i = 0; i < digits.size(); i++) {
38         val[digits[i]] = i;
39     }
40
41     for (auto digit : num) {
42         if (val[digit] >= base) {
43             return false;
44         }
45     }
46
47     return true;
48 }

```

5.6 Interactive

```

1 // you should use cout.flush() every cout
2 int query(int a) {
3     cout << "? " << a << '\n';
4     cout.flush();
5     char res; cin >> res;
6     return res;
7 }
8

```

```

9 // using endl you don't need
10 int query(int a) {
11     cout << "? " << a << endl;
12     char res; cin >> res;
13     return res;
14 }

```

5.7 Flags

```

1 // g++ -std=c++17 -Wall -Wshadow -fsanitize=address -
  02 -D -o cod a.cpp

```

5.8 Get Subsets Sum Iterative

```

1 vector<ll> get_subset_sums(int l, int r, vector<ll>&
  arr) {
2     vector<ll> ans;
3
4     int len = r-l+1;
5     for (int i = 0; i < (1 << len); i++) {
6         ll sum = 0;
7
8         for (int j = 0; j < len; j++) {
9             if (i & (1 << j)) {
10                sum += arr[l + j];
11            }
12        }
13
14        ans.push_back(sum);
15    }
16
17    return ans;
18 }

```

5.9 Last True

```

1 // Binary Search (last_true)
2
3 // last_true(2, 10, [](int x) { return x * x <= 30;
  }); // outputs 5
4 //
5 // [l, r]
6 //
7 // if none of the values in the range work, return lo
  - 1
8 //
9 // f(1) = true
10 // f(2) = true
11 // f(3) = true
12 // f(4) = true
13 // f(5) = true
14 // f(6) = false
15 // f(7) = false
16 // f(8) = false
17 //
18 // last_true(1, 8, f) = 5
19 // last_true(7, 8, f) = 6
20
21 int last_true(int lo, int hi, function<bool(int)> f)
  {
22     lo--;
23     while (lo < hi) {
24         int mid = lo + (hi - lo + 1) / 2;
25
26         if (f(mid)) {
27             lo = mid;
28         } else {
29             hi = mid - 1;
30         }
31     }
32     return lo;
33 }

```

5.10 Xor 1 To N

```

1 // XOR sum from 1 to N
2 ll xor_1_to_n(ll n) {
3     if (n % 4 == 0) {
4         return n;
5     } else if (n % 4 == 1) {
6         return 1;
7     } else if (n % 4 == 2) {
8         return n + 1;
9     }
10
11     return 0;
12 }

```

5.11 Custom Unordered Map

```

1 // Source: Tiagosf00
2
3 struct custom_hash {
4     static uint64_t splitmix64(uint64_t x) {
5         // http://xorshift.di.unimi.it/splitmix64.c
6         x += 0x9e3779b97f4a7c15;
7         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
8         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
9         return x ^ (x >> 31);
10    }
11
12    size_t operator()(uint64_t x) const {
13        static const uint64_t FIXED_RANDOM = chrono::
14        steady_clock::now().time_since_epoch().count();
15        return splitmix64(x + FIXED_RANDOM);
16    }
17 };
18 unordered_map<long long, int, custom_hash> safe_map;
19
20 // when using pairs
21 struct custom_hash {
22     inline size_t operator()(const pii & a) const {
23         return (a.first << 6) ^ (a.first >> 2) ^
24         2038074743 ^ a.second;
25     }
26 };

```

5.12 Input By File

```

1 freopen("file.in", "r", stdin);
2 freopen("file.out", "w", stdout);

```

5.13 Mix Hash

```

1 // magic hash function using mix
2
3 using ull = unsigned long long;
4 ull mix(ull o){
5     o+=0x9e3779b97f4a7c15;
6     o=(o^(o>>30))*0xbf58476d1ce4e5b9;
7     o=(o^(o>>27))*0x94d049bb133111eb;
8     return o^(o>>31);
9 }
10 ull hash(pii a) {return mix(a.first ^ mix(a.second))
11 ;}

```

5.14 Template

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main() {

```

```

6     ios::sync_with_stdio(false);
7     cin.tie(NULL);
8
9
10
11     return 0;
12 }

```

5.15 Overflow

```

1 // Signatures of some built-in functions to perform
2 // arithmetic operations with overflow check
3 // Source: https://gcc.gnu.org/onlinedocs/gcc/Integer
4 // -Overflow-Builtins.html
5 //
6 // you can also check overflow by performing the
7 // operation with double
8 // and checking if the result it's greater than the
9 // maximum value supported by the variable
10
11 bool __builtin_add_overflow (type1 a, type2 b, type3
12 *res)
13 bool __builtin_sadd_overflow (int a, int b, int *res)
14 bool __builtin_saddl_overflow (long int a, long int b
15 , long int *res)
16 bool __builtin_saddll_overflow (long long int a, long
17 long int b, long long int *res)
18 bool __builtin_uadd_overflow (unsigned int a,
19 unsigned int b, unsigned int *res)
20 bool __builtin_uaddl_overflow (unsigned long int a,
21 unsigned long int b, unsigned long int *res)
22 bool __builtin_uaddll_overflow (unsigned long long
23 int a, unsigned long long int b, unsigned long
24 long int *res)
25
26 bool __builtin_sub_overflow (type1 a, type2 b, type3
27 *res)
28 bool __builtin_ssub_overflow (int a, int b, int *res)
29 bool __builtin_ssubl_overflow (long int a, long int b
30 , long int *res)
31 bool __builtin_ssubll_overflow (long long int a, long
32 long int b, long long int *res)
33 bool __builtin_usub_overflow (unsigned int a,
34 unsigned int b, unsigned int *res)
35 bool __builtin_usubl_overflow (unsigned long int a,
36 unsigned long int b, unsigned long int *res)
37 bool __builtin_usubll_overflow (unsigned long long
38 int a, unsigned long long int b, unsigned long
39 long int *res)
40
41 bool __builtin_mul_overflow (type1 a, type2 b, type3
42 *res)
43 bool __builtin_smul_overflow (int a, int b, int *res)
44 bool __builtin_smull_overflow (long int a, long int b
45 , long int *res)
46 bool __builtin_smulll_overflow (long long int a, long
47 long int b, long long int *res)
48 bool __builtin_umul_overflow (unsigned int a,
49 unsigned int b, unsigned int *res)
50 bool __builtin_umull_overflow (unsigned long int a,
51 unsigned long int b, unsigned long int *res)
52 bool __builtin_umulll_overflow (unsigned long long
53 int a, unsigned long long int b, unsigned long
54 long int *res)
55
56 bool __builtin_add_overflow_p (type1 a, type2 b,
57 type3 c)
58 bool __builtin_sub_overflow_p (type1 a, type2 b,
59 type3 c)
60 bool __builtin_mul_overflow_p (type1 a, type2 b,
61 type3 c)

```

5.16 First True

```

1 // Binary Search (first_true)
2 //
3 // first_true(2, 10, [](int x) { return x * x >= 30;
4 // }); // outputs 6
5 //
6 // [l, r]
7 // if none of the values in the range work, return hi
8 // + 1
9 //
10 // f(4) = false
11 // f(5) = false
12 // f(6) = true
13 // f(7) = true
14 int first_true(int lo, int hi, function<bool(int)> f)
15 {
16     hi++;
17     while (lo < hi) {
18         int mid = lo + (hi - lo) / 2;
19
20         if (f(mid)) {
21             hi = mid;
22         } else {
23             lo = mid + 1;
24         }
25     }
26     return lo;

```

5.17 Xor Basis

```

1 // XOR Basis
2 // You are given a set of $N$ integer values. You
3 // should find the minimum number of values that you
4 // need to add to the set such that the following
5 // will hold true:
6 // For every two integers $A$ and $B$ in the set,
7 // their bitwise xor $A \oplus B$ is also in the set
8 //
9 vector<ll> basis;
10
11 void add(ll x) {
12     for (int i = 0; i < (int)basis.size(); i++) {
13         // reduce x using the current basis vectors
14         x = min(x, x ^ basis[i]);
15     }
16     if (x != 0) { basis.push_back(x); }
17 }
18
19 ll res = (1LL << (int)basis.size()) - n;

```

6 String

6.1 Split

```

1 vector<string> split(string s, char key=' ') {
2     vector<string> ans;
3     string aux = "";
4
5     for (int i = 0; i < (int)s.size(); i++) {
6         if (s[i] == key) {
7             if (aux.size() > 0) {
8                 ans.push_back(aux);
9                 aux = "";
10            }
11            } else {

```

```

12         aux += s[i];
13     }
14 }
15
16 if ((int)aux.size() > 0) {
17     ans.push_back(aux);
18 }
19
20 return ans;
21 }

```

6.2 Hash

```

1 struct Hash {
2     ll MOD, P;
3     int n; string s;
4     vector<ll> h, hi, p;
5     Hash() {}
6     Hash(string s, ll MOD, ll P = 31): s(s), MOD(MOD),
7     P(P), n(s.size()), h(n), hi(n), p(n) {
8         for (int i=0; i<n; i++) p[i] = (i ? P*p[i-1]:1)
9         % MOD;
10        for (int i=0; i<n; i++)
11            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
12        for (int i=n-1; i>=0; i--)
13            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
14            % MOD;
15    }
16    int query(int l, int r) {
17        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]:0)%MOD :
18        0));
19        return hash < 0 ? hash + MOD : hash;
20    }
21    int query_inv(int l, int r) {
22        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
23        +1] % MOD : 0));
24        return hash < 0 ? hash + MOD : hash;
25    }
26 };
27
28 struct DoubleHash {
29     const ll MOD1 = 90264469;
30     const ll MOD2 = 25699183;
31
32     Hash hash1, hash2;
33
34     DoubleHash();
35
36     DoubleHash(string s) : hash1(s, MOD1), hash2(s,
37     MOD2) {}
38
39     pair<int, int> query(int l, int r) {
40         return { hash1.query(l, r), hash2.query(l, r)
41         };
42     }
43
44     pair<int, int> query_inv(int l, int r) {
45         return { hash1.query_inv(l, r), hash2.
46         query_inv(l, r) };
47     }
48 };
49
50 struct TripleHash {
51     const ll MOD1 = 90264469;
52     const ll MOD2 = 25699183;
53     const ll MOD3 = 81249169;
54
55     Hash hash1, hash2, hash3;
56
57     TripleHash();
58
59     TripleHash(string s) : hash1(s, MOD1), hash2(s,
60     MOD2), hash3(s, MOD3) {}

```

```

52 tuple<int, int, int> query(int l, int r) {
53     return { hash1.query(l, r), hash2.query(l, r)
54     , hash3.query(l, r) };
55 }
56
57 tuple<int, int, int> query_inv(int l, int r) {
58     return { hash1.query_inv(l, r), hash2.
59     query_inv(l, r), hash3.query_inv(l, r) };
60 };
61
62 struct HashK {
63     vector<ll> primes; // more primes = more hashes
64     vector<Hash> hash;
65
66     HashK();
67
68     HashK(string s, vector<ll> primes): primes(primes
69 ) {
70         for (auto p : primes) {
71             hash.push_back(Hash(s, p));
72         }
73
74     vector<int> query(int l, int r) {
75         vector<int> ans;
76
77         for (auto h : hash) {
78             ans.push_back(h.query(l, r));
79         }
80
81         return ans;
82     }
83
84     vector<int> query_inv(int l, int r) {
85         vector<int> ans;
86
87         for (auto h : hash) {
88             ans.push_back(h.query_inv(l, r));
89         }
90
91         return ans;
92     }
93 };

```

6.3 Is Substring

```

1 // equivalente ao in do python
2
3 bool is_substring(string a, string b){ // verifica se
4     a Ã substring de b
5     for(int i = 0; i < b.size(); i++){
6         int it = i, jt = 0; // b[it], a[jt]
7
8         while(it < b.size() && jt < a.size()){
9             if(b[it] != a[jt])
10                 break;
11
12             it++;
13             jt++;
14
15             if(jt == a.size())
16                 return true;
17         }
18
19         return false;
20 }

```

6.4 Trie Xor

```

1 // TrieXOR
2 //
3 // adiciona, remove e verifica se existe strings
4 // binarias
5 // max_xor(x) = maximiza o xor de x com algum valor
6 // da trie
7 //
8 // raiz = 0
9 //
10 // https://codeforces.com/problemset/problem/706/D
11 //
12 // 0(|s|) adicionar, remover e buscar
13
14 struct TrieXOR {
15     int n, alph_sz, nxt;
16     vector<vector<int>> trie;
17     vector<int> finish, paths;
18
19     TrieXOR() {}
20
21     TrieXOR(int n, int alph_sz = 2) : n(n), alph_sz(
22     alph_sz) {
23         nxt = 1;
24         trie.assign(n, vector<int>(alph_sz));
25         finish.assign(n * alph_sz, 0);
26         paths.assign(n * alph_sz, 0);
27     }
28
29     void add(int x) {
30         int curr = 0;
31
32         for (int i = 31; i >= 0; i--) {
33             int b = ((x << i) > 0);
34
35             if (trie[curr][b] == 0)
36                 trie[curr][b] = nxt++;
37
38             paths[curr]++;
39             curr = trie[curr][b];
40         }
41
42         finish[curr]++;
43     }
44
45     void rem(int x) {
46         int curr = 0;
47
48         for (int i = 31; i >= 0; i--) {
49             int b = ((x << i) > 0);
50
51             paths[curr]--;
52             curr = trie[curr][b];
53         }
54
55         finish[curr]--;
56     }
57
58     int search(int x) {
59         int curr = 0;
60
61         for (int i = 31; i >= 0; i--) {
62             int b = ((x << i) > 0);
63
64             if (trie[curr][b] == 0) return false;
65
66             curr = trie[curr][b];
67         }
68
69         return (finish[curr] > 0);
70     }
71 }

```

```

71 int max_xor(int x) { // maximum xor with x and
    any number of trie
72     int curr = 0, ans = 0;
73
74     for (int i = 31; i >= 0; i--) {
75         int b = ((x & (1 << i)) > 0);
76         int want = b^1;
77
78         if (trie[curr][want] == 0 || paths[trie[
curr][want]] == 0) want ^= 1;
79         if (trie[curr][want] == 0 || paths[trie[
curr][want]] == 0) break;
80         if (want != b) ans |= (1 << i);
81
82         curr = trie[curr][want];
83     }
84
85     return ans;
86 }
87 };

```

7 DP

7.1 Digit Dp

```

1 // Digit DP 1: https://atcoder.jp/contests/dp/tasks/
  dp_s
2 //
3 // find the number of integers between 1 and K (
  inclusive)
4 // where the sum of digits in base ten is a multiple
  of D
5
6 #include <bits/stdc++.h>
7
8 using namespace std;
9
10 const int MOD = 1e9+7;
11
12 string k;
13 int d;
14
15 int tb[10010][110][2];
16
17 int dp(int pos, int sum, bool under) {
18     if (pos >= k.size()) return sum == 0;
19
20     int& mem = tb[pos][sum][under];
21     if (mem != -1) return mem;
22     mem = 0;
23
24     int limit = 9;
25     if (!under) limit = k[pos] - '0';
26
27     for (int digit = 0; digit <= limit; digit++) {
28         mem += dp(pos+1, (sum + digit) % d, under | (
digit < limit));
29         mem %= MOD;
30     }
31
32     return mem;
33 }
34
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(NULL);
38
39     cin >> k >> d;
40
41     memset(tb, -1, sizeof(tb));
42

```

```

43     cout << (dp(0, 0, false) - 1 + MOD) % MOD << '\n'
    ;
44
45     return 0;
46 }

```

7.2 Lcs

```

1 // LCS (Longest Common Subsequence)
2 //
3 // maior subsequencia comum entre duas strings
4 //
5 // tamanho da matriz da dp eh |a| x |b|
6 // lcs(a, b) = string da melhor resposta
7 // dp[a.size()][b.size()] = tamanho da melhor
  resposta
8 //
9 // https://atcoder.jp/contests/dp/tasks/dp_f
10 //
11 // O(n^2)
12
13 string lcs(string a, string b) {
14     int n = a.size();
15     int m = b.size();
16
17     int dp[n+1][m+1];
18     pair<int, int> p[n+1][m+1];
19
20     memset(dp, 0, sizeof(dp));
21     memset(p, -1, sizeof(p));
22
23     for (int i = 1; i <= n; i++) {
24         for (int j = 1; j <= m; j++) {
25             if (a[i-1] == b[j-1]) {
26                 dp[i][j] = dp[i-1][j-1] + 1;
27                 p[i][j] = {i-1, j-1};
28             } else {
29                 if (dp[i-1][j] > dp[i][j-1]) {
30                     dp[i][j] = dp[i-1][j];
31                     p[i][j] = {i-1, j};
32                 } else {
33                     dp[i][j] = dp[i][j-1];
34                     p[i][j] = {i, j-1};
35                 }
36             }
37         }
38     }
39
40     // recuperar resposta
41
42     string ans = "";
43     pair<int, int> curr = {n, m};
44
45     while (curr.first != 0 && curr.second != 0) {
46         auto [i, j] = curr;
47
48         if (a[i-1] == b[j-1]) {
49             ans += a[i-1];
50         }
51
52         curr = p[i][j];
53     }
54
55     reverse(ans.begin(), ans.end());
56
57     return ans;
58 }

```

7.3 Lis Binary Search

```

1 int lis(vector<int> arr) {
2     vector<int> dp;

```



```

3
4     for (auto e : arr) {
5         int pos = lower_bound(dp.begin(), dp.end(), e
6     ) - dp.begin();
7
8         if (pos == (int)dp.size()) {
9             dp.push_back(e);
10        } else {
11            dp[pos] = e;
12        }
13    }
14
15    return (int)dp.size();
16 }

```

7.4 Knapsack

```

1 //Submeter em c++ 64bits otimiza o long long
2 ll knapsack(vector<ll>& weight, vector<ll>& value,
3     int W) {
4     //Usar essa knapsack se sã§ precisar do resultado
5     final.
6     //O(W) em memã§ria
7     vector<vector<ll>> table(2, vector<ll>(W + 1, 0))
8     ;
9     int n = (int)value.size();
10
11    for(int k = 1; k <= n; k++) {
12        for(int i = 0; i <= W; i++) {
13            if(i - weight[k - 1] >= 0) {
14                table[k % 2][i] = max(table[(k - 1)
15                % 2][i],
16                value[k - 1] + table[(k - 1) %
17                2][i - weight[k - 1]]);
18            } else {
19                table[k % 2][i] = max(table[(k - 1) %
20                2][i], table[k % 2][i]);
21            }
22        }
23    }
24
25    return table[n % 2][W];
26 }
27
28 ll knapsack(vector<ll>& weight, vector<ll>& value,
29     int W) {
30     //Usar essa knapsack se, em algum momento,
31     precisar recuperar os indices
32     //O(NW) em memã§ria
33
34     int n = (int)value.size();
35     vector<vector<ll>> table(W + 1, vector<ll>(n + 1,
36     0));
37
38     for(int k = 1; k <= n; k++) {
39         for(int i = 0; i <= W; i++) {
40             if(i - weight[k - 1] >= 0) {
41                 table[i][k] = max(table[i][k - 1],
42                 value[k - 1] + table[i - weight[k -
43                 1]][k - 1]);
44             } else {
45                 table[i][k] = max(table[i][k - 1],
46                 table[i][k]);
47             }
48         }
49     }
50
51     /*
52     int per = W;
53     vector<int> idx;
54     for(int k = n; k > 0; k--) {
55         if(table[per][k] == table[per][k - 1]){
56             continue;

```

```

46         } else {
47             idx.push_back(k - 1);
48             per -= weight[k - 1];
49         }
50     }
51     */
52
53     return table[W][n];
54 }
55
56 const int MOD = 998244353;
57
58 struct Knapsack {
59
60     int S; // max value
61     vector<ll> dp;
62
63     Knapsack(int S_) {
64         S = S_ + 5;
65         dp.assign(S, 0);
66         dp[0] = 1;
67     }
68
69     void Add(int val) {
70         if(val <= 0 || val >= S) return;
71         for(int i = S - 1; i >= val; i--) {
72             dp[i] += dp[i - val];
73             dp[i] %= MOD;
74         }
75     }
76
77     void Rem(int val) {
78         if(val <= 0 || val >= S) return;
79         for(int i = val; i < S; i++) {
80             dp[i] += MOD - dp[i - val];
81             dp[i] %= MOD;
82         }
83     }
84
85     int Query(int val) {
86         // # of ways to select a subset of numbers
87         with sum = val
88         if(val <= 0 || val >= S) return 0;
89         return dp[val];
90     }
91 };
92
93 void solve() {
94     int n, w;
95     cin >> n >> w;
96     vector<ll> weight(n), value(n);
97     for(int i = 0; i < n; i++) {
98         cin >> weight[i] >> value[i];
99     }
100     cout << knapsack(weight, value, w) << "\n";
101 }
102
103
104 }

```

7.5 Edit Distance

```

1 // Edit Distance / Levenshtein Distance
2 //
3 // numero minimo de operacoes
4 // para transformar
5 // uma string em outra
6 //
7 // tamanho da matriz da dp eh |a| x |b|
8 // edit_distance(a.size(), b.size(), a, b)
9 //
10 // https://cses.fi/problemset/task/1639

```

```

11 //
12 // 0(n^2)
13
14 int tb[MAX][MAX];
15
16 int edit_distance(int i, int j, string &a, string &b)
17 {
18     if (i == 0) return j;
19     if (j == 0) return i;
20
21     int &ans = tb[i][j];
22
23     if (ans != -1) return ans;
24
25     ans = min({
26         edit_distance(i-1, j, a, b) + 1,
27         edit_distance(i, j-1, a, b) + 1,
28         edit_distance(i-1, j-1, a, b) + (a[i-1] != b[
29             j-1])
30     });
31
32     return ans;
33 }

```

7.6 Digit Dp 2

```

1 // Digit DP 2: https://cses.fi/problemset/task/2220
2 //
3 // Number of integers between a and b
4 // where no two adjacent digits are the same
5
6 #include <bits/stdc++.h>
7
8 using namespace std;
9 using ll = long long;
10
11 const int MAX = 20; // 10^18
12
13 ll tb[MAX][MAX][2][2];
14
15 ll dp(string& number, int pos, int last_digit, bool
16     under, bool started) {
17     if (pos >= (int)number.size()) {
18         return 1;
19     }
20
21     ll& mem = tb[pos][last_digit][under][started];
22     if (mem != -1) return mem;
23     mem = 0;
24
25     int limit = 9;
26     if (!under) limit = number[pos] - '0';
27
28     for (int digit = 0; digit <= limit; digit++) {
29         if (started && digit == last_digit) continue;
30
31         bool is_under = under || (digit < limit);
32         bool is_started = started || (digit != 0);
33
34         mem += dp(number, pos+1, digit, is_under,
35             is_started);
36     }
37
38     return mem;
39 }
40
41 ll solve(ll ubound) {
42     memset(tb, -1, sizeof(tb));
43     string number = to_string(ubound);
44     return dp(number, 0, 10, 0, 0);
45 }
46
47 int main() {

```

```

46     ios::sync_with_stdio(false);
47     cin.tie(NULL);
48
49     ll a, b; cin >> a >> b;
50     cout << solve(b) - solve(a-1) << '\n';
51
52     return 0;
53 }

```

7.7 Lis Segtree

```

1 int n, arr[MAX], aux[MAX]; cin >> n;
2 for (int i = 0; i < n; i++) {
3     cin >> arr[i];
4     aux[i] = arr[i];
5 }
6
7 sort(aux, aux+n);
8
9 Segtree st(n); // seg of maximum
10
11 int ans = 0;
12 for (int i = 0; i < n; i++) {
13     int it = lower_bound(aux, aux+n, arr[i]) - aux;
14     int lis = st.query(0, it) + 1;
15
16     st.update(it, lis);
17
18     ans = max(ans, lis);
19 }
20
21 cout << ans << '\n';

```

7.8 Range Dp

```

1 // Range DP 1: https://codeforces.com/problemset/
2 // problem/1132/F
3 //
4 // You may apply some operations to this string
5 // in one operation you can delete some contiguous
6 // substring of this string
7 // if all letters in the substring you delete are
8 // equal
9 // calculate the minimum number of operations to
10 // delete the whole string s
11
12 #include <bits/stdc++.h>
13
14 using namespace std;
15
16 const int MAX = 510;
17
18 int n, tb[MAX][MAX];
19 string s;
20
21 int dp(int left, int right) {
22     if (left > right) return 0;
23
24     int& mem = tb[left][right];
25     if (mem != -1) return mem;
26
27     mem = 1 + dp(left+1, right); // gastar uma
28     // opera~o arrumando sã o cara atual
29     for (int i = left+1; i <= right; i++) {
30         if (s[left] == s[i]) {
31             mem = min(mem, dp(left+1, i-1) + dp(i,
32                 right));
33         }
34     }
35
36     return mem;
37 }

```

```

32
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(NULL);
36
37     cin >> n >> s;
38     memset(tb, -1, sizeof(tb));
39     cout << dp(0, n-1) << '\n';
40
41     return 0;
42 }

```

8 DS

8.1 Range Color Update

```

1 // Range color update (brunomaletta)
2 //
3 // update(l, r, c) colore o range [l, r] com a cor c,
4 // e retorna os ranges que foram coloridos [l, r, cor]
5 // query(i) retorna a cor da posicao i
6 //
7 // Complexidades (para q operacoes):
8 // update - O(log(q)) amortizado
9 // query - O(log(q))
10
11 template<typename T> struct color {
12     set<tuple<int, int, T>> se;
13
14     vector<tuple<int, int, T>> update(int l, int r, T
15     val) {
16         auto it = se.upper_bound({r, INF, val});
17         if (it != se.begin() and get<1>(*prev(it)) >
18         r) {
19             auto [L, R, V] = *--it;
20             se.erase(it);
21             se.emplace(L, r, V), se.emplace(r+1, R, V
22         );
23         }
24         it = se.lower_bound({l, -INF, val});
25         if (it != se.begin() and get<1>(*prev(it)) >=
26         l) {
27             auto [L, R, V] = *--it;
28             se.erase(it);
29             se.emplace(L, l-1, V), it = se.emplace(l,
30             R, V).first;
31         }
32         vector<tuple<int, int, T>> ret;
33         for (; it != se.end() and get<0>(*it) <= r;
34         it = se.erase(it))
35             ret.push_back(*it);
36         se.emplace(l, r, val);
37         return ret;
38     }
39
40     T query(int i) {
41         auto it = se.upper_bound({i, INF, T()});
42         if (it == se.begin() or get<1>(*--it) < i)
43             return -1; // nao tem
44         return get<2>(*it);
45     }
46 };

```

8.2 Trie Old

```

1 struct Trie {
2
3     int nxt = 1, sz, maxLet = 26; //tamanho do
4     alfabeto
5     vector< vector<int> > trie;
6     bitset<(int)1e7> finish; //modificar esse valor
7     pra ser >= n

```

```

6 //garantir que vai submeter em cpp 64
7
8 Trie(int n){
9     sz = n;
10    trie.assign(sz, vector<int>(maxLet,0));
11 }
12
13 void add(string &s){
14     int cur = 0;
15     for(auto c: s){
16         //alterar esse azinho dependendo da
17         entrada!!
18         if(trie[cur][c-'a'] == 0){
19             trie[cur][c-'a'] = nxt++;
20             cur = trie[cur][c-'a'];
21         } else {
22             cur = trie[cur][c-'a'];
23         }
24     }
25     finish[cur] = 1;
26 }
27
28 int search(string& s){
29     int cur = 0;
30     for(auto c: s){
31         if(trie[cur][c-'a'] == 0){
32             return 0;
33         }
34         cur = trie[cur][c-'a'];
35     }
36     return finish[cur];
37 }
38 };

```

8.3 Treap

```

1 // treap CP algo
2
3 typedef struct item * pitem;
4 struct item {
5     int prior, value, cnt;
6     bool rev;
7     pitem l, r;
8 };
9
10 int cnt (pitem it) {
11     return it ? it->cnt : 0;
12 }
13
14 void upd_cnt (pitem it) {
15     if (it)
16         it->cnt = cnt(it->l) + cnt(it->r) + 1;
17 }
18
19 void push (pitem it) {
20     if (it && it->rev) {
21         it->rev = false;
22         swap (it->l, it->r);
23         if (it->l) it->l->rev ^= true;
24         if (it->r) it->r->rev ^= true;
25     }
26 }
27
28 void merge (pitem & t, pitem l, pitem r) {
29     push (l);
30     push (r);
31     if (!l || !r)
32         t = l ? l : r;
33     else if (l->prior > r->prior)
34         merge (l->r, l->l, r), t = l;
35     else
36         merge (r->l, l, r->r), t = r;

```

```

37     upd_cnt (t);
38 }
39
40 void split (pitem t, pitem & l, pitem & r, int key,
41             int add = 0) {
42     if (!t)
43         return void( l = r = 0 );
44     push (t);
45     int cur_key = add + cnt(t->l);
46     if (key <= cur_key)
47         split (t->l, l, t->l, key, add), r = t;
48     else
49         split (t->r, t->r, r, key, add + 1 + cnt(t->l
50     )), l = t;
51     upd_cnt (t);
52 }
53
54 void reverse (pitem t, int l, int r) {
55     pitem t1, t2, t3;
56     split (t, t1, t2, l);
57     split (t2, t2, t3, r-l+1);
58     t2->rev ^= true;
59     merge (t, t1, t2);
60     merge (t, t, t3);
61 }
62
63 void output (pitem t) {
64     if (!t) return;
65     push (t);
66     output (t->l);
67     printf ("%d ", t->value);
68     output (t->r);
69 }

```

8.4 Sparse

```

1 struct Sparse {
2
3     vector<vector<int>> arr;
4
5     int op(int& a, int& b){ //min, max, gcd, lcm, and
6         , or
7         return min(a,b);
8         //return __gcd(a,b);
9         //return max(a,b);
10    }
11
12    Sparse(vector<int>& v){ //Constrói a tabela
13        int n = v.size(), logn = 0;
14        while((1<<logn) <= n) logn++;
15        arr.assign(n, vector<int>(logn, 0));
16        for(int i = 0; i < n; i++){
17            arr[i][0] = v[i];
18            for(int k = 1; k < logn; k++){
19                for(int i = 0; i < n; i++){
20                    if(i + ( 1 << k) -1 >= n)
21                        break;
22                    int p = i+( 1 << (k-1) );
23                    arr[i][k] = op( arr[i][ k-1 ] , arr[p
24                ][k-1] );
25            }
26        }
27
28        int query(int l, int r){
29            int pot = 31 - __builtin_clz(r-l+1); //r-l+1
30            sÃo INTEIROS, nÃo ll
31            int k = (1 << pot) ;
32            return op( arr[l][pot] , arr[ r - (k-1) ][
33                pot] );
34        }
35    };

```

8.5 Mex

```

1 // Mex
2 //
3 // facilita queries de mex com update
4 //
5 // N eh o maior valor possÃvel do mex
6 // add(x) = adiciona x
7 // rem(x) = remove x
8 //
9 // O(log N) por insert
10 // O(1) por query
11
12 struct Mex {
13     map<int, int> cnt;
14     set<int> possible;
15
16     Mex(int n) {
17         for (int i = 0; i <= n + 1; i++) {
18             possible.insert(i);
19         }
20     }
21
22     void add(int x) {
23         cnt[x]++;
24         possible.erase(x);
25     }
26
27     void rem(int x) {
28         cnt[x]--;
29
30         if (cnt[x] == 0) {
31             possible.insert(x);
32         }
33     }
34
35     int query() {
36         return *(possible.begin());
37     }
38 };

```

8.6 Bit

```

1 struct BIT {
2     int n, LOGN = 0;
3     vector<ll> bit;
4
5     BIT(int nn){
6         n = nn + 10;
7         bit.resize(n + 10, 0);
8         while( (1LL << LOGN) <= n ) LOGN++;
9     }
10
11     ll query(int x){
12         x++;
13         ll ans = 0;
14         while(x > 0){
15             ans += bit[x];
16             x -= (x & (-x));
17         }
18         return ans;
19     }
20
21     void update(int x, ll val){
22         x++;
23         while(x < (int)bit.size()){
24             bit[x] += val;
25             x += (x & (-x));
26         }
27     }
28
29     int findkth(int k){
30         //kth smallest, 0(logN)

```

```

31 //use position i to count how many times
value 'i' appear
32 int sum = 0, pos = 0;
33 for(int i = LOGN; i >= 0; i--){
34     if(pos + (1LL << i) < n && sum + bit[pos
+ (1LL << i)] < k){
35         sum += bit[pos + (1LL << i)];
36         pos += (1LL << i);
37     }
38 }
39 return pos;
40 }
41 /*
42 int findkth(int k){
43     //kth smallest, O(log^2(N))
44     //use position i to count how many times
value 'i' appear
45     int x = 0, mx = 200;
46     for(int b = n; b > 0 && mx > 0; b /= 2){
47         while( x+b < n && query(x+b) < k && mx--
> 0 ){
48             x += b;
49         }
50     }
51     return x+1;
52 }
53 */
54 };

```

8.7 Maxqueue

```

1 struct MaxQueue {
2     stack< pair<ll,ll> > in, out;
3
4     void add(ll x){
5         if(in.size())
6             in.push( { x, max(x, in.top().ss) } );
7         else
8             in.push( {x, x} );
9     }
10
11 ll get_max(){
12     if(in.size() > 0 && out.size() > 0)
13         return max(in.top().ss, out.top().ss);
14     else if(in.size() > 0) return in.top().ss;
15     else if(out.size() > 0) return out.top().ss;
16     else return INF;
17 }
18
19
20
21 void rem(){
22
23     if(out.size() == 0){
24         while(in.size()){
25             ll temp = in.top().ff, ma;
26             if(out.size() == 0) ma = temp;
27             else ma = max(temp, out.top().ss);
28             out.push({temp, ma});
29             in.pop();
30         }
31     }
32     //removendo o topo de out
33     out.pop();
34 }
35
36 ll size(){
37     return in.size() + out.size();
38 }
39
40 };

```

8.8 Dsu

```

1 // DSU
2 //
3 // https://judge.yosupo.jp/submission/126864
4
5 struct DSU {
6     int n = 0, components = 0;
7     vector<int> parent;
8     vector<int> size;
9
10     DSU(int nn){
11         n = nn;
12         components = n;
13         size.assign(n + 5, 1);
14         parent.assign(n + 5, 0);
15         iota(parent.begin(), parent.end(), 0);
16     }
17
18     int find(int x){
19         if(x == parent[x]) {
20             return x;
21         }
22         //path compression
23         return parent[x] = find(parent[x]);
24     }
25
26     void join(int a, int b){
27         a = find(a);
28         b = find(b);
29
30         if(a == b) {
31             return;
32         }
33
34         if(size[a] < size[b]) {
35             swap(a, b);
36         }
37
38         parent[b] = a;
39         size[a] += size[b];
40         components -= 1;
41     }
42
43     int sameSet(int a, int b) {
44         a = find(a);
45         b = find(b);
46         return a == b;
47     }
48 };

```

8.9 Segtree

```

1 struct Segtree {
2
3     int n; //size do array que a seg vai ser criada
em cima
4     vector<ll> seg;
5
6     Segtree(vector<ll>& s){
7         n = (int)s.size();
8         seg.resize(n+n+n, 0);
9         seg_build(1,0,n-1,s);
10    }
11
12    ll merge(ll a, ll b){
13        //return a+b;
14        if(!a) a = 00;
15        if(!b) b = 00;
16        return min(a,b);
17    }
18

```

```

19 void seg_build(int x, int l, int r, vector<ll>& s) {
20     if(r < l) return;
21     if(l == r){
22         seg[x] = s[l];
23     } else {
24         int mid = l + (r-l)/2;
25         seg_build(x+x, l, mid, s);
26         seg_build(x+x+1, mid+1, r, s);
27         seg[x] = merge(seg[x+x], seg[x+x+1]);
28     }
29 }
30
31 //atual, intervalo na Árvore e intervalo
32 //pedido
33 ll q(int x, int l, int r, int i, int j){
34     if(r < i || l > j) return 0;
35     if(l >= i && r <= j) return seg[x];
36     int mid = l + (r-l)/2;
37     return merge(q(x+x, l, mid, i, j), q(x+x+1, mid+1,
38     r, i, j));
39 }
40
41 //att posi pra val
42 void att(int x, int l, int r, int posi, ll val){
43     if(l == r){
44         seg[x] = val;
45     } else {
46         int mid = l + (r-l)/2;
47         if(posi <= mid) att(x+x, l, mid, posi, val);
48         else att(x+x+1, mid+1, r, posi, val);
49         seg[x] = merge(seg[x+x], seg[x+x+1]);
50     }
51 }
52
53 int findkth(int x, int l, int r, int k){
54     if(l == r){
55         return l;
56     } else {
57         int mid = l + (r-l)/2;
58         if(seg[x+x] >= k){
59             return findkth(x+x, l, mid, k);
60         } else {
61             return findkth(x+x+1, mid+1, r, k -
62             seg[x+x]);
63         }
64     }
65 }
66
67 ll query(int l, int r){
68     return q(1, 0, n-1, l, r);
69 }
70
71 void update(int posi, ll val){ //alterar em posi
72     pra val
73     att(1, 0, n-1, posi, val);
74 }
75
76 int findkth(int k){
77     //kth smallest, O(logN)
78     //use position i to count how many times
79     value 'i' appear
80     //merge must be the sum of nodes
81     return findkth(1, 0, n-1, k);
82 }
83
84 };
85
86 8.10 Seglazystuctnode
87
88 struct Node {
89     int l, r;
90     int pref0, suf0, best0;
91     int pref1, suf1, best1;
92
93     Node(){
94         pref0 = 0; suf0 = 0; best0 = 0;
95         pref1 = 0; suf1 = 0; best1 = 0;
96         l = -1; r = -1;
97     };
98
99     void Init(int val_, int l_, int r_) {
100         best0 = !val_;
101         pref0 = !val_;
102         suf0 = !val_;
103
104         best1 = val_;
105         pref1 = val_;
106         suf1 = val_;
107
108         l = l_;
109         r = r_;
110     }
111
112     bool AllZero() {
113         return r - l + 1 == best0;
114     }
115
116     bool AllOne() {
117         return r - l + 1 == best1;
118     }
119
120     void Reverse() {
121         swap(pref0, pref1);
122         swap(suf0, suf1);
123         swap(best0, best1);
124     }
125
126     Node Merge(Node a, Node b) {
127         if(a.l == -1 || a.r == -1) {
128             return b;
129         }
130
131         if(b.l == -1 || b.r == -1) {
132             return a;
133         }
134
135         auto ans = Node();
136
137         ans.l = a.l;
138         ans.r = b.r;
139
140         // -----
141         //
142
143         if(a.AllZero()) {
144             ans.pref0 = a.pref0 + b.pref0;
145         } else {
146             ans.pref0 = a.pref0;
147         }
148
149         if(b.AllZero()) {
150             ans.suf0 = b.suf0 + a.suf0;
151         } else {
152             ans.suf0 = b.suf0;
153         }
154
155         ans.best0 = max({
156             a.best0,

```

```

76         b.best0,
77         a.suf0 + b.pref0
78     });
79
80     // -----
81     //
82
83     if(a.AllOne()) {
84         ans.pref1 = a.pref1 + b.pref1;
85     } else {
86         ans.pref1 = a.pref1;
87     }
88
89     if(b.AllOne()) {
90         ans.suf1 = b.suf1 + a.suf1;
91     } else {
92         ans.suf1 = b.suf1;
93     }
94
95     ans.best1 = max({
96         a.best1,
97         b.best1,
98         a.suf1 + b.pref1
99     });
100
101     // -----
102     //
103     return ans;
104 }
105
106 struct SegLazy {
107     private:
108
109         int n;
110         vector<Node> seg;
111         vector<bool> lazy; // precisa reverter ou nao
112
113         void build(ll x, int l, int r, string& s){
114             if(l == r){
115                 int val = s[l] - '0';
116                 seg[x].Init(val, l, r);
117             } else {
118                 int mid = l + (r-l)/2;
119                 build(x+x, l, mid, s);
120                 build(x+x+1, mid+1, r, s);
121                 seg[x] = Merge(seg[x+x], seg[x+x+1]);
122             }
123         }
124
125     void upd_lazy(ll node, ll l, ll r){
126
127         if(lazy[node]) {
128             seg[node].Reverse();
129         }
130
131         ll esq = node + node, dir = esq + 1;
132
133         if(dir < (int)seg.size() && lazy[node]){
134             lazy[esq] = !lazy[esq];
135             lazy[dir] = !lazy[dir];
136         }
137
138         lazy[node] = 0;
139     }
140
141     Node q(ll x, int l, int r, int i, int j){
142         upd_lazy(x,l,r);
143
144         if(r < i || l > j)
145             return Node();
146
147         if(l >= i && r <= j )
148             return seg[x];
149
150         int mid = l + (r-l)/2;
151         return Merge(q(x+x,l,mid,i,j), q(x+x+1,
152             mid+1,r,i,j));
153     }
154
155     void upd(ll x, int l, int r, int i, int j){
156         upd_lazy(x,l,r);
157         if(r < i || l > j) return;
158         if(l >= i && r <= j){
159             lazy[x] = !lazy[x];
160             upd_lazy(x,l,r);
161         } else {
162             int mid = l + (r-l)/2;
163             upd(x+x,l,mid,i,j);
164             upd(x+x+1,mid+1,r,i,j);
165             seg[x] = Merge(seg[x+x], seg[x+x+1]);
166         }
167     }
168
169     public:
170
171     SegLazy(string& s){
172         n = (int)s.size();
173         seg.assign(n+n+n+n, Node());
174         lazy.assign(n+n+n+n, 0);
175         build(1,0,n-1,s);
176     }
177
178     void update(int l){
179         upd(1,0,n-1,l,l);
180     }
181
182     void update_range(int l, int r){
183         upd(1,0,n-1,l,r);
184     }
185
186     Node query(int l){
187         return q(1, 0, n-1, l, l);
188     }
189
190     Node query(int l, int r){
191         return q(1, 0, n-1, l, r);
192     }
193
194     void solve() {
195         int n, q;
196         string s;
197
198         cin >> n >> q >> s;
199
200         SegLazy seg(s);
201
202         while(q--) {
203             int c, l, r;
204             cin >> c >> l >> r;
205
206             if(c == 1) {
207                 // inverte l...r
208                 seg.update_range(l - 1, r - 1);
209             } else {
210                 // query l...r
211                 auto node = seg.query(l - 1, r - 1);

```

```

219         cout << node.best1 << "\n";
220     }
221 }
222 }
223 }
224 }

```

8.11 Mergesorttree

```

1 //const int MAXN = 3e5 + 10;
2 //vector<int> seg[ 4 * MAXN + 10];
3
4 struct MergeSortTree {
5
6     int n; //size do array que a seg vai ser criada
7     em cima
8     vector< vector<int> > seg;
9     //vector< vector<ll> > ps; //prefix sum
10
11     MergeSortTree(vector<int>& s){
12         //se o input for grande (ou o tempo mt puxado
13         ), coloca a seg com size
14         //maximo de forma global
15         n = (int)s.size();
16         seg.resize(4 * n + 10);
17         //ps.resize(4 * n + 10);
18         seg_build(1,0,n-1,s);
19     }
20
21     vector<int> merge(vi& a, vi& b){
22         int i = 0, j = 0, p = 0;
23         vi ans(a.size() + b.size());
24         while(i < (int)a.size() && j < (int)b.size()){
25
26             if(a[i] < b[j]){
27                 ans[p++] = a[i++];
28             } else {
29                 ans[p++] = b[j++];
30             }
31         }
32         while(i < (int)a.size()){
33             ans[p++] = a[i++];
34         }
35         while(j < (int)b.size()){
36             ans[p++] = b[j++];
37         }
38         return ans;
39     }
40
41     vector<ll> calc(vi& s) {
42         ll sum = 0;
43         vector<ll> tmp;
44         for(auto &x : s) {
45             sum += x;
46             tmp.push_back(sum);
47         }
48         return tmp;
49     }
50
51     void seg_build(int x, int l, int r, vector<int>&
52     s){
53         if(r < l) return;
54         if(l == r){
55             seg[x].push_back(s[l]);
56             //ps[x] = {s[l]};
57         } else {
58             int mid = l + (r-l)/2;
59             seg_build(x+x, l, mid, s);
60             seg_build(x+x+1, mid+1, r, s);
61             seg[x] = merge(seg[x+x], seg[x+x+1]);
62             //ps[x] = calc(seg[x]);
63         }
64     }
65 }

```

```

61
62 //nÃs atual, intervalo na Ãrvore e intervalo
63 pedido
64 // retorna a quantidade de numeros <= val em [l,
65 r]
66
67 ll q(int x, int l, int r, int i, int j, int val){
68     if(r < i || l > j) return 0;
69     if(l >= i && r <= j){
70         return (lower_bound(seg[x].begin(), seg[x]
71         .end(), val) - seg[x].begin());
72     }
73     int mid = l + (r-l)/2;
74     return q(x+x,l,mid,i,j, val) + q(x+x+1,mid+1,
75     r,i,j, val);
76 }
77
78 // retorna a soma dos numeros <= val em [l, r]
79 // nÃs atual, intervalo na Ãrvore e intervalo
80 pedido
81 /*
82 ll q(int x, int l, int r, int i, int j, ll val){
83     if(r < i || l > j) return 0;
84     if(l >= i && r <= j){
85         auto it = upper_bound(seg[x].begin(), seg
86         [x].end(), val) - seg[x].begin();
87
88         if(val > seg[x].back()) {
89             return ps[x].back();
90         }
91
92         if(val < seg[x][0]) {
93             return 0;
94         }
95
96         return ps[x][it - 1];
97     }
98
99     int mid = l + (r-l)/2;
100     return q(x+x,l,mid,i,j, val) + q(x+x+1,mid+1,
101     r,i,j, val);
102 }
103 */
104 ll query(int l, int r, ll val){
105     return q(1, 0, n-1, l, r, val);
106 }
107
108 };

```

8.12 Seghash

```

1 template<typename T> //use as SegtreeHash<int> h or
2 SegtreeHash<char>
3 struct SegtreeHash {
4
5     int n; //size do array que a seg vai ser criada
6     em cima
7
8     // P = 31, 53, 59, 73 .... (prime > number of
9     different characters)
10    // M = 578398229, 895201859, 1e9 + 7, 1e9 + 9 (
11    big prime)
12    int p, m;
13
14    vector<ll> seg, pot;
15
16    ll minValue = 0; // menor valor possÃvel que
17    pode estar na estrutura
18    // isso Ã pra evitar que a hash
19    de '0' seja igual a de '0000...'

```



```

14 SegtreeHash(vector<T>& s, ll P = 31, ll MOD = (1ll
15 )1e9 + 7){
16     n = (int)s.size();
17     p = P; m = MOD;
18     seg.resize(4 * n, -1);
19     pot.resize(4 * n);
20     pot[0] = 1;
21     for(int i = 1; i < (int)pot.size(); i++) {
22         pot[i] = (pot[i - 1] * P) % MOD;
23     }
24     seg_build(1, 0, n - 1, s);
25 }
26
27 ll merge(ll a, ll b, int tam){
28     if(a == -1) return b;
29     if(b == -1) return a;
30     return (a + b * pot[tam]) % m;
31 }
32
33 void seg_build(int x, int l, int r, vector<T>& s)
34 {
35     if(r < l) return;
36     if(l == r){
37         seg[x] = (int)s[l] - minValue + 1;
38     } else {
39         int mid = l + (r-l)/2;
40         seg_build(x+x, l, mid, s);
41         seg_build(x+x+1, mid+1, r, s);
42         seg[x] = merge(seg[x+x], seg[x+x+1], mid
43 - l + 1);
44     }
45
46     //nÃo atual, intervalo na Ãrvore e intervalo
47     pedido
48     ll q(int x, int l, int r, int i, int j){
49         if(r < i || l > j) return -1;
50         if(l >= i && r <= j) return seg[x];
51         int mid = l + (r-l)/2;
52         return merge(q(x+x,l,mid,i,j), q(x+x+1,mid+1,
53 r,i,j), mid - max(i, l) + 1);
54     }
55
56     //att posi pra val
57     void att(int x, int l, int r, int posi, T val){
58         if(l == r){
59             seg[x] = (int)val - minValue + 1;
60         } else {
61             int mid = l + (r-l)/2;
62             if(posi <= mid) att(x+x,l,mid,posi,val);
63             else att(x+x+1,mid+1,r,posi,val);
64             seg[x] = merge(seg[x+x], seg[x+x+1], mid
65 - l + 1);
66         }
67     }
68
69     ll query(int l, int r){
70         return q(1, 0, n-1, l, r);
71     }
72
73     void update(int posi, T val){ //alterar em posi
74         pra val
75         att(1, 0, n-1, posi, val);
76     }
77 };
78
79 8.13 Segtree Lazy Iterative
80
81 // Segtree iterativa com lazy
82 //
83 // https://codeforces.com/gym/103708/problem/C
84
85 //
86 // 0(N * log(N)) build
87 // 0(log(N)) update e query
88
89 const int MAX = 524288; // NEED TO BE POWER OF 2 !!!
90 const int LOG = 19; // LOG = ceil(log2(MAX))
91
92 namespace seg {
93     ll seg[2*MAX], lazy[2*MAX];
94     int n;
95
96     ll junta(ll a, ll b) {
97         return a+b;
98     }
99
100     // soma x na posicao p de tamanho tam
101     void poe(int p, ll x, int tam, bool prop=1) {
102         seg[p] += x*tam;
103         if (prop and p < n) lazy[p] += x;
104     }
105
106     // atualiza todos os pais da folha p
107     void sobe(int p) {
108         for (int tam = 2; p /= 2; tam *= 2) {
109             seg[p] = junta(seg[2*p], seg[2*p+1]);
110             poe(p, lazy[p], tam, 0);
111         }
112     }
113
114     void upd_lazy(int i, int tam) {
115         if (lazy[i] && (2 * i + 1) < 2 * MAX) {
116             poe(2*i, lazy[i], tam);
117             poe(2*i+1, lazy[i], tam);
118             lazy[i] = 0;
119         }
120     }
121
122     // propaga o caminho da raiz ate a folha p
123     void prop(int p) {
124         int tam = 1 << (LOG-1);
125         for (int s = LOG; s; s--, tam /= 2) {
126             int i = p >> s;
127             upd_lazy(i, tam);
128         }
129     }
130
131     void build(int n2) {
132         n = n2;
133         for (int i = 0; i < n; i++) seg[n+i] = 0;
134         for (int i = n-1; i; i--) seg[i] = junta(seg
135 [2*i], seg[2*i+1]);
136         for (int i = 0; i < 2*n; i++) lazy[i] = 0;
137     }
138
139     ll query(int a, int b) {
140         ll ret = 0;
141         for (prop(a+=n), prop(b+=n); a <= b; ++a/=2,
142 --b/=2) {
143             if (a%2 == 1) ret = junta(ret, seg[a]);
144             if (b%2 == 0) ret = junta(ret, seg[b]);
145         }
146         return ret;
147     }
148
149     void update(int a, int b, int x) {
150         int a2 = a += n, b2 = b += n, tam = 1;
151         for (; a <= b; ++a/=2, --b/=2, tam *= 2) {
152             if (a%2 == 1) poe(a, x, tam);
153             if (b%2 == 0) poe(b, x, tam);
154         }
155         sobe(a2), sobe(b2);
156     }
157 }

```

```

75 int findkth(int x, int l, int r, ll k, int tam){
76     int esq = x + x;
77     int dir = x + x + 1;
78
79     upd_lazy(x, tam);
80     upd_lazy(esq, tam/2);
81     upd_lazy(dir, tam/2);
82
83     if(l == r){
84         return l;
85     } else {
86         int mid = l + (r-1)/2;
87
88         if(seg[esq] >= k){
89             return findkth(esq, l, mid, k, tam/2);
90         } else {
91             return findkth(dir, mid+1, r, k - seg[
92 esq], tam/2);
93         }
94     }
95
96     int findkth(ll k){
97         // kth smallest, O(logN)
98         // use position i to count how many times
99         // value 'i' appear
100         // merge must be the sum of nodes
101         return findkth(1, 0, n-1, k, (1 << (LOG-1)));
102     };

```

8.14 Seglazy

```

1 struct SegLazy {
2
3     int n;
4     vector<ll> seg;
5     vector<ll> lazy;
6
7     SegLazy(vector<ll>& arr){
8         n = (int)arr.size();
9         seg.assign(n+n+n+n, 0);
10        lazy.assign(n+n+n+n, 0);
11        build(1, 0, n-1, arr);
12    }
13
14    ll merge(ll a, ll b){
15        return a+b;
16    }
17
18    void build(ll x, int l, int r, vector<ll>& arr){
19        if(l == r){
20            seg[x] = 1LL * arr[l];
21        } else {
22            int mid = l + (r-1)/2;
23            build(x+x, l, mid, arr);
24            build(x+x+1, mid+1, r, arr);
25            seg[x] = merge(seg[x+x], seg[x+x+1]);
26        }
27    }
28
29    void upd_lazy(ll node, ll l, ll r){
30        seg[node] += (ll)(r-l+1) * lazy[node];
31        ll esq = node + node, dir = esq + 1;
32
33        if(dir < (int)seg.size()){
34            lazy[esq] += lazy[node];
35            lazy[dir] += lazy[node];
36        }
37
38        lazy[node] = 0;
39    }
40

```

```

41 ll q(ll x, int l, int r, int i, int j){
42     upd_lazy(x, l, r);
43
44     if(r < i || l > j)
45         return 0;
46
47     if(l >= i && r <= j )
48         return seg[x];
49
50     int mid = l + (r-1)/2;
51     return merge(q(x+x, l, mid, i, j), q(x+x+1, mid+1,
52 r, i, j));
53 }
54
55 ll query(int l, int r){ //valor em uma posi
56     //especifica -> query de [l, l];
57     return q(1, 0, n-1, l, r);
58 }
59
60 void upd(ll x, int l, int r, int i, int j, ll u){
61     upd_lazy(x, l, r);
62     if(r < i || l > j) return;
63     if(l >= i && r <= j){
64         lazy[x] += u;
65         upd_lazy(x, l, r);
66     } else {
67         int mid = l + (r-1)/2;
68         upd(x+x, l, mid, i, j, u);
69         upd(x+x+1, mid+1, r, i, j, u);
70         seg[x] = merge(seg[x+x], seg[x+x+1]);
71     }
72 }
73
74 void upd_range(int l, int r, ll u){ //intervalo e
75     //valor
76     upd(1, 0, n-1, l, r, u);
77 }

```

8.15 Bit2d

```

1 struct BIT2D {
2
3     int n, m;
4     vector<vector<int>>> bit;
5
6     BIT2D(int nn, int mm) {
7         //use as 0-indexed, but inside here I will
8         //use 1-indexed positions
9         n = nn + 2;
10        m = mm + 2;
11        bit.assign(n, vector<int>(m));
12    }
13
14    void update(int x, int y, int p) {
15        x++; y++;
16        assert(x > 0 && y > 0 && x <= n && y <= m);
17        for(; x < n; x += (x & (-x)))
18            for(int j = y; j < m; j += (j & (-j)))
19                bit[x][j] += p;
20    }
21
22    int sum(int x, int y) {
23        int ans = 0;
24        for(; x > 0; x -= (x & (-x)))
25            for(int j = y; j > 0; j -= (j & (-j)))
26                ans += bit[x][j];
27        return ans;
28    }
29
30    int query(int x, int y, int p, int q) {
31        //x...p on line, y...q on column

```

```

31 //sum from [x][y] to [p][q];
32 x++; y++; p++; q++;
33 assert(x > 0 && y > 0 && x <= n && y <= m);
34 assert(p > 0 && q > 0 && p <= n && q <= m);
35 return sum(p, q) - sum(x - 1, q) - sum(p, y -
1) + sum(x - 1, y - 1);
36 }
37
38
39 };

```

8.16 Ordered Set

```

1 // Ordered Set
2 //
3 // set roubado com mais operacoes
4 //
5 // para alterar para multiset
6 // trocar less para less_equal
7 //
8 // ordered_set<int> s
9 //
10 // order_of_key(k) // number of items strictly
// smaller than k -> int
11 // find_by_order(k) // k-th element in a set (
// counting from zero) -> iterator
12 //
13 // https://cses.fi/problemset/task/2169
14 //
15 // O(log N) para insert, erase (com iterator),
// order_of_key, find_by_order
16
17 using namespace __gnu_pbds;
18 template <typename T>
19 using ordered_set = tree<T,null_type,less<T>,
// rb_tree_tag,tree_order_statistics_node_update>;
20
21 void erase(ordered_set& a, int x){
22     int r = a.order_of_key(x);
23     auto it = a.find_by_order(r);
24     a.erase(it);
25 }

```

8.17 Cht

```

1 // CHT (tiagodfs)
2
3 const ll is_query = -LLINF;
4 struct Line{
5     ll m, b;
6     mutable function<const Line*> succ;
7     bool operator<(const Line& rhs) const{
8         if(rhs.b != is_query) return m < rhs.m;
9         const Line* s = succ();
10        if(!s) return 0;
11        ll x = rhs.m;
12        return b - s->b < (s->m - m) * x;
13    }
14 };
15 struct Cht : public multiset<Line>{ // maintain max m
// *x+b
16     bool bad(iterator y){
17         auto z = next(y);
18         if(y == begin()){
19             if(z == end()) return 0;
20             return y->m == z->m && y->b <= z->b;
21         }
22         auto x = prev(y);
23         if(z == end()) return y->m == x->m && y->b <=
x->b;
24         return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)
(y->b - z->b)*(y->m - x->m);

```

```

25 }
26 void insert_line(ll m, ll b){ // min -> insert (-
m,-b) -> -eval()
27     auto y = insert({ m, b });
28     y->succ = [=]{ return next(y) == end() ? 0 :
&*next(y); };
29     if(bad(y)){ erase(y); return; }
30     while(next(y) != end() && bad(next(y))) erase
(next(y));
31     while(y != begin() && bad(prev(y))) erase(
prev(y));
32 }
33 ll eval(ll x){
34     auto l = *lower_bound((Line) { x, is_query })
;
35     return l.m * x + l.b;
36 }
37 };

```

8.18 Bigk

```

1 struct SetSum {
2     ll sum;
3     multiset<ll> ms;
4
5     SetSum() {}
6
7     void add(ll x) {
8         sum += x;
9         ms.insert(x);
10    }
11
12    int rem(ll x) {
13        auto it = ms.find(x);
14
15        if (it == ms.end()) {
16            return 0;
17        }
18
19        sum -= x;
20        ms.erase(it);
21        return 1;
22    }
23
24    ll getMin() { return *ms.begin(); }
25
26    ll getMax() { return *ms.rbegin(); }
27
28    ll getSum() { return sum; }
29
30    int size() { return (int)ms.size(); }
31 };
32
33 struct BigK {
34     int k;
35     SetSum gt, mt;
36
37     BigK(int k): k(k) {}
38
39     void balance() {
40         while (gt.size() > k) {
41             ll mn = gt.getMin();
42             gt.rem(mn);
43             mt.add(mn);
44         }
45
46         while (gt.size() < k && mt.size() > 0) {
47             ll mx = mt.getMax();
48             mt.rem(mx);
49             gt.add(mx);
50         }
51     }
52 }

```

```

53 void add(ll x) {
54     gt.add(x);
55     balance();
56 }
57
58 void rem(ll x) {
59     if (mt.rem(x) == 0) {
60         gt.rem(x);
61     }
62
63     balance();
64 }
65
66 // be careful, O(abs(oldK - newk) * log)
67 void setK(int _k) {
68     k = _k;
69     balance();
70 }
71
72 // O(log)
73 void incK() { setK(k + 1); }
74
75 // O(log)
76 void decK() { setK(k - 1); }
77 };

```

8.19 Querytree

```

1 struct QueryTree {
2     int n, t = 0, l = 3, build = 0, euler = 0;
3     vector<ll> dist;
4     vector<int> in, out, d;
5     vector<vector<int>> sobe;
6     vector<vector<pair<int, ll>>> arr;
7     vector<vector<ll>> table_max; //max edge
8     vector<vector<ll>> table_min; //min edge
9
10    QueryTree(int nn) {
11        n = nn + 5;
12        arr.resize(n);
13        in.resize(n);
14        out.resize(n);
15        d.resize(n);
16        dist.resize(n);
17        while( (1 << l) < n ) l++;
18        sobe.assign(n + 5, vector<int>(++l));
19        table_max.assign(n + 5, vector<ll>(l));
20        table_min.assign(n + 5, vector<ll>(l));
21    }
22
23    void add_edge(int u, int v, ll w){ //
24        bidirectional edge with weight w
25        arr[u].push_back({v, w});
26        arr[v].push_back({u, w});
27    }
28
29    //assert the root of tree is node 1 or change the
30    'last' in the next function
31    void Euler_Tour(int u, int last = 1, ll we = 0,
32    int depth = 0, ll sum = 0){ //euler tour
33        euler = 1; //remember to use this function
34        before the queries
35        in[u] = t++;
36        d[u] = depth;
37        dist[u] = sum; //sum = sum of the values in
38        edges from root to node u
39        sobe[u][0] = last; //parent of u. parent of 1
40        is 1
41        table_max[u][0] = we;
42        table_min[u][0] = we;
43        for(auto v: arr[u]) if(v.ff != last){
44            Euler_Tour(v.ff, u, v.ss, depth + 1, sum
45            + v.ss);

```

```

39    }
40    out[u] = t++;
41    }
42
43    void build_table(){ //binary lifting
44        assert(euler);
45        build = 1; //remeber use this function before
46        queries
47        for(int k = 1; k < l; k++){
48            for(int i = 1; i <= n; i++){
49                sobe[i][k] = sobe[sobe[i][k-1]][k-1];
50                table_max[i][k] = max(table_max[i][k
51                - 1], table_max[sobe[i][k-1]][k-1]);
52                table_min[i][k] = min(table_min[i][k
53                - 1], table_min[sobe[i][k-1]][k-1]);
54            }
55        }
56
57        int is_ancestor(int u, int v){ // return 1 if u
58        is ancestor of v
59        assert(euler);
60        return in[u] <= in[v] && out[u] >= out[v];
61    }
62
63    int lca(int u, int v){ //return lca of u and v
64    assert(build && euler);
65    if(is_ancestor(u,v)) return u;
66    if(is_ancestor(v,u)) return v;
67    int lca = u;
68    for(int k = l - 1; k >= 0; k--){
69        int tmp = sobe[lca][k];
70        if(!is_ancestor(tmp, v)){
71            lca = tmp;
72        }
73    }
74    return sobe[lca][0];
75
76    int lca(int u, int v, int root) { //return lca of
77    u and v when tree is rooted at 'root'
78    return lca(u, v) ^ lca(v, root) ^ lca(root, u
79    ); //magic
80    }
81
82    int up_k(int u, int qt){ //return node k levels
83    higher starting from u
84    assert(build && euler);
85    for(int b = 0; b < l; b++){
86        if(qt%2) u = sobe[u][b];
87        qt >>= 1;
88    }
89    return u;
90
91    ll goUpMax(int u, int to){ //return the max
92    weigth of a edge going from u to 'to'
93    assert(build);
94    if(u == to) return 0;
95    ll mx = table_max[u][0];
96    for(int k = l - 1; k >= 0; k--){
97        int tmp = sobe[u][k];
98        if( !is_ancestor(tmp, to) ){
99            mx = max(mx, table_max[u][k]);
100            u = tmp;
101        }
102    }
103    return max(mx, table_max[u][0]);
104
105    ll max_edge(int u, int v){ //return the max
106    weight of a edge in the simple path from u to v
107    assert(build);

```

```

103     int ancestor = lca(u, v);
104     ll a = goUpMax(u, ancestor), b = goUpMax(v,
ancestor);
105     if(ancestor == u) return b;
106     else if(ancestor == v) return a;
107     return max(a,b);
108 }
109
110 ll goUpMin(int u, int to){ //return the min
weight of a edge going from u to 'to'
111     assert(build);
112     if(u == to) return oo;
113     ll mx = table_min[u][0];
114     for(int k = 1 - 1; k >= 0; k--){
115         int tmp = sobe[u][k];
116         if( !is_ancestor(tmp, to) ){
117             mx = min(mx, table_min[u][k]);
118             u = tmp;
119         }
120     }
121     return min(mx, table_min[u][0]);
122 }
123
124 ll min_edge(int u, int v){ //return the min
weight of a edge in the simple path from u to v
125     assert(build);
126     int ancestor = lca(u, v);
127     ll a = goUpMin(u, ancestor), b = goUpMin(v,
ancestor);
128     if(ancestor == u) return b;
129     else if(ancestor == v) return a;
130     return min(a,b);
131 }
132
133 ll query_dist(int u, int v){ //distance of nodes
u and v
134     int x = lca(u, v);
135     return dist[u] - dist[x] + dist[v] - dist[x];
136 }
137
138 int kth_between(int u, int v, int k){ //kth node
in the simple path from u to v; if k = 1, ans = u
139     k--;
140     int x = lca(u, v);
141     if( k > d[u] - d[x] ){
142         k -= (d[u] - d[x]);
143         return up_k(v, d[v]-d[x]-k);
144     }
145     return up_k(u, k);
146 }
147
148 };
149
150 int main() {
151     ios::sync_with_stdio(false);
152     cin.tie(NULL);
153
154     int t = 1, n, u, v, w, k;
155     string s;
156     cin >> t;
157     while(t--){
158         cin >> n;
159         QueryTree arr(n);
160         for(int i = 1; i < n; i++){
161             cin >> u >> v >> w;
162             arr.add_edge(u,v,w);
163         }
164         arr.Euler_Tour(1);
165         arr.build_table();
166         while(cin >> s, s != "DONE"){
167             cin >> u >> v;
168             if(s == "DIST") {
169                 cout << arr.query_dist(u, v) << "\n";

```

```

170     } else {
171         cin >> k;
172         cout << arr.kth_between(u,v,k) << "\n
";
173     }
174 }
175     cout << "\n";
176 }
177 }
178 }

```

8.20 Manhattan Mst

```

1 /**
2  * Author: chilli, Takanori MAEHARA
3  * Date: 2019-11-02
4  * License: CC0
5  * Source: https://github.com/spaghetti-source/
algorithm/blob/master/geometry/rectilinear_mst.cc
6  * Description: Given N points, returns up to 4*N
edges, which are guaranteed
7  * to contain a minimum spanning tree for the graph
with edge weights  $w(p, q) =$ 
8  *  $|p.x - q.x| + |p.y - q.y|$ . Edges are in the form (
distance, src, dst). Use a
9  * standard MST algorithm on the result to find the
final MST.
10 * Time:  $O(N \log N)$ 
11 * Status: Stress-tested
12 */
13 /**
14 * Author: Ulf Lundstrom
15 * Date: 2009-02-26
16 * License: CC0
17 * Source: My head with inspiration from tinyKACTL
18 * Description: Class to handle points in the plane.
19 * T can be e.g. double or long long. (Avoid int.)
20 * Status: Works fine, used a lot
21 */
22
23 #pragma once
24
25 template <class T> int sgn(T x) { return (x > 0) - (x
< 0); }
26
27 template<class T>
28 struct Point {
29     typedef Point P;
30     T x, y;
31     explicit Point(T x=0, T y=0) : x(x), y(y) {}
32     bool operator<(P p) const { return tie(x,y) < tie
(p.x,p.y); }
33     bool operator==(P p) const { return tie(x,y)==tie
(p.x,p.y); }
34     P operator+(P p) const { return P(x+p.x, y+p.y);
}
35     P operator-(P p) const { return P(x-p.x, y-p.y);
}
36     P operator*(T d) const { return P(x*d, y*d); }
37     P operator/(T d) const { return P(x/d, y/d); }
38     T dot(P p) const { return x*p.x + y*p.y; }
39     T cross(P p) const { return x*p.y - y*p.x; }
40     T cross(P a, P b) const { return (a-*this).cross(
b-*this); }
41     T dist2() const { return x*x + y*y; }
42     double dist() const { return sqrt((double)dist2()
); }
43     // angle to x-axis in interval [-pi, pi]
44     double angle() const { return atan2(y, x); }
45     P unit() const { return *this/dist(); } // makes
dist()==1
46     P perp() const { return P(-y, x); } // rotates
+90 degrees
47     P normal() const { return perp().unit(); }

```

```

47 // returns point rotated 'a' radians ccw around
   the origin
48 P rotate(double a) const {
49     return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a));
   };
50 friend ostream& operator<<(ostream& os, P p) {
51     return os << "(" << p.x << "," << p.y << ")";
   };
52 };
53
54 typedef Point<int> P;
55 vector<array<int, 3>> manhattanMST(vector<P> ps) {
56     vi id(sz(ps));
57     iota(all(id), 0);
58     vector<array<int, 3>> edges;
59     rep(k,0,4) {
60         sort(all(id), [&](int i, int j) {
61             return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y
62         });
63         map<int, int> sweep;
64         for (int i : id) {
65             for (auto it = sweep.lower_bound(-ps[i].y
66         );
67             it != sweep.end(); sweep.
68         erase(it++)) {
69             int j = it->second;
70             P d = ps[i] - ps[j];
71             if (d.y > d.x) break;
72             edges.push_back({d.y + d.x, i, j});
73             sweep[-ps[i].y] = i;
74         }
75         for (P& p : ps) if (k & 1) p.x = -p.x; else
76         swap(p.x, p.y);
77     }
78     return edges;
79 }

```

8.21 Trie

```

1 struct Trie {
2
3     struct Node {
4         map<char, Node> adj; // dÃ pra trocar por
5         vector(26)
6         ll finishHere;
7
8         Node() {
9             finishHere = 0;
10         }
11
12         bool find(char c) {
13             return adj.find(c) != adj.end();
14         }
15     };
16
17     Node mainNode;
18
19     Trie(){
20         mainNode = Node();
21     }
22
23     void add(string &s) {
24         Node *curNode = &mainNode;
25
26         for(auto &c : s) {
27
28             if(!curNode->find(c)) {
29                 curNode->adj[c] = Node();
30             }
31
32             curNode = &curNode->adj[c];

```

```

   }
   curNode->finishHere += 1;
}

void dfs(Node& node) {
    for(auto &v : node.adj) {
        dfs(v.ss);
        // faz alguma coisa
    }
}

void dfs() {
    return dfs(mainNode);
}

bool search(string &s) {
    Node* curNode = &mainNode;

    for(auto &c : s) {
        if(!curNode->find(c))
            return false;

        curNode = &curNode->adj[c];
    }

    return curNode->finishHere > 0;
}

void debugRec(Node node, int depth) {
    for(auto &x : node.adj) {
        cout << string(3 * depth, ' ') << x.ff <<
        " " << x.ss.finishHere << "\n";
        debugRec(x.ss, depth + 1);
    }
}

void debug() {
    debugRec(mainNode, 0);
}
};

```

8.22 Triexor

```

1 struct Trie {
2
3     int nxt = 1, sz, maxLet = 2;
4     vector< vector<int> > trie;
5     vector<int> finish, paths;
6
7     Trie(int n){
8         sz = n;
9         trie.assign(sz + 10, vector<int>(maxLet,0));
10        finish.resize(sz + 10);
11        paths.resize(sz+10);
12    }
13
14    void add(int x){
15        int cur = 0;
16        for(int i = 31; i >= 0; i--){
17            int b = ( (x & (1 << i)) > 0);
18            if(trie[cur][b] == 0)
19                trie[cur][b] = nxt++;
20            cur = trie[cur][b];
21            paths[cur]++;
22        }
23        paths[cur]++;
24    }
25
26    void rem(int x){
27        int cur = 0;
28        for(int i = 31; i >= 0; i--){

```

```

29         int b = ( (x & (1 << i)) > 0);
30         cur = trie[cur][b];
31         paths[cur]--;
32     }
33     finish[cur]--;
34     paths[cur]--;
35 }
36
37 int query(int x){ //return the max xor with x
38     int ans = 0, cur = 0;
39
40     for(int i = 31; i >= 0; i--){
41         int b = ( (x & (1 << i)) > 0);
42         int bz = trie[cur][0];
43         int bo = trie[cur][1];
44
45         if(bz > 0 && bo > 0 && paths[bz] > 0 &&
46            paths[bo] > 0){
47             //cout << "Optimal" << endl;
48             cur = trie[cur][b ^ 1];
49             ans += (1 << i);
50         } else if(bz > 0 && paths[bz] > 0){
51             //cout << "Zero" << endl;
52             cur = trie[cur][0];
53             if(b) ans += (1 << i);
54         } else if(bo > 0 && paths[bo] > 0){
55             //cout << "One" << endl;
56             cur = trie[cur][1];
57             if(!b) ans += (1 << i);
58         } else {
59             break;
60         }
61     }
62     return ans;
63 }
64
65 };

```

8.23 Kruskal

```

1 struct Edge {
2     int u, v;
3     ll weight;
4
5     Edge() {}
6
7     Edge(int u, int v, ll weight) : u(u), v(v),
8     weight(weight) {}
9
10    bool operator<(Edge const& other) {
11        return weight < other.weight;
12    }
13 };
14
15 vector<Edge> kruskal(vector<Edge> edges, int n) {
16     vector<Edge> result;

```

```

16     ll cost = 0;
17
18     sort(edges.begin(), edges.end());
19     DSU dsu(n);
20
21     for (auto e : edges) {
22         if (!dsu.same(e.u, e.v)) {
23             cost += e.weight;
24             result.push_back(e);
25             dsu.unite(e.u, e.v);
26         }
27     }
28
29     return result;
30 }

```

8.24 Psum2d

```

1 struct PSum {
2
3     vector<vi> arr;
4     int n, m, initialized = 0;
5
6     PSum(int _n, int _m) {
7         n = _n;
8         m = _m;
9         arr.resize(n + 2);
10        arr.assign(n + 2, vector<int>(m + 2, 0));
11    }
12
13    void add(int a, int b, int c) {
14        //a and b are 0-indexed
15        arr[a + 1][b + 1] += c;
16    }
17
18    void init() {
19        for(int i = 1; i <= n; i++) {
20            for(int j = 1; j <= m; j++) {
21                arr[i][j] += arr[i][j - 1];
22                arr[i][j] += arr[i - 1][j];
23                arr[i][j] -= arr[i - 1][j - 1];
24            }
25        }
26        initialized = 1;
27    }
28
29    int query(int a, int b, int c, int d) {
30        // sum of a...c and b...d
31        // a, b, c and d are 0-indexed
32        assert(initialized);
33        return arr[c + 1][d + 1] - arr[a][d + 1] -
34        arr[c + 1][b] + arr[a][b];
35    }
36 };

```