teambrbr002
UFMG

Emanuel Silva, Felipe Mota e Kaio Vieira

22 de agosto de 2025

Índice

1 Graph

1.1 2SAT

```
d9d struct TwoSat {
060   int N;
67e   vector<vector<int>> E;
662   TwoSat(int N) : N(N), E(2 * N) {}
11c   int neg(int u) const {
46c     return (u + N) % (2 * N);
9ac   }
b0e   void add_or(int u, int v) {
c7f     E[neg(u)].push_back(v);
1e6     E[neg(v)].push_back(u);
120   }
4b9   void add_nand(int u, int v) {
0f2     E[u].push_back(neg(v));
28d     E[v].push_back(neg(u));
a1e   }
f78   void add_true(int u) {
708     E[neg(u)].emplace_back(u);
29a   }
fec   void add_not(int u) {
27d     add_true(neg(u));
668   }
4a5   void add_xor(int u, int v) {
ef7     add_or(u, v);
fd0     add_nand(u, v);
a32   }
15a   void add_and(int u, int v) {
3ec     add_true(u);
ca9     add_true(v);
0ca   }
c52   void add_nor(int u, int v) {
01f     add_and(neg(u), neg(v));
465   }
e75   void add_xnor(int u, int v) {
5c6     add_xor(u, neg(v));
bc8   }
      // Assumes tarjan sorts SCCs in reverse topological order (u -> v
          implies scc[v] <= scc[u]).
1f8   pair<bool, vector<bool>> solve() const {
a78     vector<bool> res(N);
58d     auto scc = tarjan(E);
fd1     for (int u = 0; u < N; ++u) {
938       if (scc[u] == scc[neg(u)]) return {false, {}};
fcc       res[u] = scc[neg(u)] > scc[u];
fd7     }
39e     return pair(true, res);
8d5   }
```

```
c83 };
```

1.2 Binary Lifting

```
24f template<const int LOG_N>
e38 struct BinaryLifting {
c2f   int timer;
361   vector<int> tin, tout, h, up[LOG_N];
903   vector<vector<int>> adj;
741   BinaryLifting(int N) : timer(0), tin(N), tout(N), h(N), adj(N) {
eb5     for(int j = 0; j < LOG_N; ++j) {
085       up[j].assign(N, 0);
751     }
952   }
58b   void add_edge(int u, int v) {
166     adj[u].emplace_back(v);
281     adj[v].emplace_back(u);
f1f   }
1bb   void set_root(int root) {
043     dfs(root, root);
f6c   }
fb6   void dfs(int u, int p) {
406     tin[u] = timer++;
f5f     up[0][u] = p;
4c7     for(int i = 0; i + 1 < LOG_N; ++i) {
07e       up[i + 1][u] = up[i][up[i][u]];
e2b     }
d1c     for(auto v : adj[u]) {
40d       if(v == p) {
5e2         continue;
05a       }
41f       h[v] = h[u] + 1;
95e       dfs(v, u);
734     }
4f8     tout[u] = timer;
286   }
42d   bool is_ancestor(int u, int v) const {
b6f     return tin[u] <= tin[v] && tout[u] >= tout[v];
320   }
0d7   int lca(int u, int v) const {
a47     if(is_ancestor(u, v)) {
03f       return u;
625     }
de8     if(is_ancestor(v, u)) {
6dc       return v;
3b5     }
b4e     for(int i = LOG_N - 1; i >= 0; --i) {
86b       if(!is_ancestor(up[i][u], v)) {
b8b         u = up[i][u];
fa1       }
5d2     }
e4a     return up[0][u];
d66   }
40d   int dist(int u, int v) const {
671     return h[u] + h[v] - 2 * h[lca(u, v)];
3fa   }
e69   int go_up(int u, int steps) const {
da8     for(int i = 0; i < LOG_N; ++i) {
94c       if((steps >> i) & 1) {
b8b         u = up[i][u];
b3e       }
9f7     }
03f     return u;
350   }
043 };
```

1.3 Bridges Online

```
514 int par[MAXN], sz[MAXN], ds[MAXN], twcc[MAXN], mark[MAXN], upd_twcc[MAXN],
    mark_cnt, bridges;
940 void init(int n) {
027   iota(ds, ds + n, 0);
171   iota(twcc, twcc + n, 0);
401   fill(sz, sz + n, 1);
487   fill(par, par + n, -1);
8cd }
23f int ds_root(int x) { return ds[x] == x ? x : ds[x] = ds_root(ds[x]); }
69d int twcc_root(int x) { return twcc[x] == x ? x : twcc[x] =
    twcc_root(twcc[x]); }
930 void rootify(int u) {
f28   mark_cnt++;
a05   int root = u, lst = -1;
a72   while(u != -1) {
ef2     if(mark[twcc_root(u)] != mark_cnt) {
564       mark[twcc_root(u)] = mark_cnt;
2d6       upd_twcc[twcc_root(u)] = u;
dda     }
c6c     if(twcc_root(u) == u) {
b09       twcc[upd_twcc[u]] = upd_twcc[u];
680       twcc[u] = upd_twcc[u];
2ae     }
6c4     int nxt = par[u];
a09     par[u] = lst;
f7e     ds[u] = root;
074     lst = u;
646     u = nxt;
a57   };
6a6   sz[root] = sz[lst];
725 }
9c4 void unite_comp(int a, int b) {
7a8   int ca = ds_root(a), cb = ds_root(b);
343   if(sz[ca] < sz[cb]) {
c3e     swap(ca, cb);
257     swap(a, b);
e10   }
7cb   rootify(b);
```

```
58b   par[b] = ds[b] = a;
c69   sz[ca] += sz[b];
f20 }
549 bool check_lca(int x, vector<int>& px) {
53b   if(x != -1) {
829     px.emplace_back(x);
dfb     if(mark[x] == mark_cnt) {
8a6       return true;
d8c     }
2d7     mark[x] = mark_cnt;
38f   }
d1f   return false;
ad0 }
46c void remove_bridges(int lca, vector<int>& px) {
0aa   for(auto x : px) {
da8     twcc[x] = lca;
722     if(x == lca) break;
c8e     --bridges;
9eb   }
082 }
98b void unite_path(int a, int b) {
f28   mark_cnt++;
fb8   a = twcc_root(a), b = twcc_root(b);
9d0   vector<int> pa, pb;
dd9   int lca = -1;
1f8   while(lca == -1) {
cc4     if(a != -1) {
afa       pa.emplace_back(a);
a05       if(mark[a] == mark_cnt) {
7cf         lca = a;
c2b         break;
9d5       }
1ab       mark[a] = mark_cnt;
f50       a = par[a] == -1 ? -1 : twcc_root(par[a]);
77f     }
6a5     if(b != -1) {
e55       pb.emplace_back(b);
da9       if(mark[b] == mark_cnt) {
abb         lca = b;
c2b         break;
262       }
496       mark[b] = mark_cnt;
f90       b = par[b] == -1 ? -1 : twcc_root(par[b]);
1a9     }
28e   }
6ef   remove_bridges(lca, pa);
6a9   remove_bridges(lca, pb);
bb1 }
010 void add_edge(int a, int b) {
9c7   if(ds_root(a) != ds_root(b)) {
b4b     ++bridges;
4c6     unite_comp(a, b);
f17   } else if(twcc_root(a) != twcc_root(b)) {
6f3     unite_path(a, b);
b4d   }
d11 }
```

1.4 Centroid Decomposition

```
0b8 vector<int> adj[MAXN]; int sz[MAXN]; bool blocked[MAXN];
e34 int preprocess(int u, int p) {
267   sz[u] = 1;
d1c   for(auto v : adj[u]) {
a01     if(v == p || blocked[v]) continue;
557     sz[u] += preprocess(v, u);
f6b   }
f93   return sz[u];
5b1 }
d9b int get_centroid(int u, int p, int tree_size) {
d1c   for(auto v : adj[u]) {
a01     if(v == p || blocked[v]) continue;
cc1     if(2 * sz[v] >= tree_size) return get_centroid(v, u, tree_size);
29e   }
03f   return u;
945 }
e36 void centroid_decomposition(int u, int lst_c) {
fd1   int c = get_centroid(u, -1, preprocess(u, -1));
3dd   blocked[c] = true;
      /* do something with centroid c */
0df   for(auto v : adj[c]) {
c85     if(!blocked[v]) centroid_decomposition(v, c);
b98   }
33e }
```

1.5 Dinic

```
67a template<typename T>
14d struct Dinic {
e9b   struct Edge {
791     int to;
d90     T cap, flow;
112     Edge(int to, T cap) : to(to), cap(cap), flow(0) {}
338     T res() const { return cap - flow; }
e92   };
05b   int m = 0, n;
321   vector<Edge> edges;
903   vector<vector<int>> adj;
3b3   vector<int> dist, ptr;
ef9   Dinic(int n) : n(n), adj(n), dist(n), ptr(n) {}
555   void add_edge(int u, int v, T cap) {
df5     if(u != v) {
2b3       edges.emplace_back(v, cap);
265       edges.emplace_back(u, 0);
30f       adj[u].emplace_back(m++);
6ab       adj[v].emplace_back(m++);
```

```
296     }
a09   }
123   bool bfs(int s, int t) {
fd5     fill(begin(dist), end(dist), n + 1);
a93     dist[s] = 0;
0b4     queue<int> q({s});
14d     while(!q.empty()) {
e4a       int u = q.front();
833       q.pop();
4b5       if(u == t) break;
cfc       for(int id : adj[u]) {
815         Edge& e = edges[id];
d9e         if(e.res() > 0 && dist[e.to] > dist[u] + 1) {
29b           dist[e.to] = dist[u] + 1;
a78           q.emplace(e.to);
08b         }
9fc       }
c1c     }
8b6     return dist[t] != n + 1;
10a   }
d6a   T dfs(int u, int t, T flow) {
3b2     if(u == t || flow == 0) {
99d       return flow;
b48     }
9f1     for(int& i = ptr[u]; i < (int)adj[u].size(); ++i) {
12b       Edge& e = edges[adj[u][i]];
187       Edge& oe = edges[adj[u][i] ^ 1];
02c       if(dist[e.to] == dist[oe.to] + 1) {
4cf         T amt = min(flow, e.res());
f17         if(T ret = dfs(e.to, t, amt)) {
786           e.flow += ret;
a4c           oe.flow -= ret;
edf           return ret;
f59         }
4a5       }
d53     }
bb3     return 0;
def   }
9c4   T max_flow(int s, int t) {
c80     T total = 0;
8ce     while(bfs(s, t)) {
197       fill(begin(ptr), end(ptr), 0);
419       while(T flow = dfs(s, t, numeric_limits<T>::max())) {
810         total += flow;
136       }
70c     }
994     return total;
eb4   }
      //returns where in the min-cut (S,T) the vertex u is
      //false: u in S, true: u in T
159   bool cut(int u) const { return dist[u] == n + 1; }
e68 };
```

1.6 Euler Path - directed

```
3ac vector<pair<int, int>> adj[MAXN];
803 int ind[MAXN], outd[MAXN];
691 vector<int> path, path_edges;
cc5 void calc_deg(int n) {
19f   for(int u = 0; u < n; ++u) {
329     for(auto [v, id] : adj[u]) {
478       ind[v]++;
686       outd[u]++;
967     }
ea1   }
2ea }
65b bool has_eulerian_path(int n) {
864   int st_cnt = 0, ft_cnt = 0;
19f   for(int u = 0; u < n; ++u) {
a21     if(abs(ind[u] - outd[u]) > 1) return false;
d62     if(outd[u] - ind[u] == +1) st_cnt++;
49b     if(outd[u] - ind[u] == -1) ft_cnt++;
5a4   }
e80   return (st_cnt == 0 && ft_cnt == 0) // eulerian circuit
daf       || (st_cnt == 1 && ft_cnt == 1); // eulerian path
b6b }
733 void dfs(int u, int from_id) {
004   while(!adj[u].empty()) {
978     auto [v, id] = adj[u].back();
687     adj[u].pop_back();
bd8     dfs(v, id);
5c4   }
264   path.emplace_back(u);
d1c   if(from_id != -1) path_edges.emplace_back(from_id);
147 }
78b int get_start_node(int n) {
c1a   int st = 0;
19f   for(int u = 0; u < n; ++u) {
aa7     if(outd[u] - ind[u] == +1) return u;
d48     if(outd[u] > 0) st = u;
ddc   }
aa0   return st;
d30 }
ea5 bool get_eulerian_path(int n, int m) {
a9e   calc_deg(n);
f50   if(!has_eulerian_path(n)) return false;
b69   dfs(get_start_node(n), -1);
a31   return (int)path_edges.size() == m;
732 }
// remember to reverse path_edges!
```

1.7 Euler Path - undirected

```
3ac vector<pair<int, int>> adj[MAXN];
793 int deg[MAXN]; bool seen_edges[MAXM];
```

```
691 vector<int> path, path_edges;
cc5 void calc_deg(int n) {
19f   for(int u = 0; u < n; ++u) {
329     for(auto [v, id] : adj[u]) {
418       deg[v]++;
717     }
eea   }
da6 }
65b bool has_eulerian_path(int n) {
03c   int odd_cnt = 0;
19f   for(int u = 0; u < n; ++u) {
8ae     if(deg[u] % 2) odd_cnt++;
808   }
70c   return odd_cnt == 0 // eulerian circuit
6a8       || odd_cnt == 2; // eulerian path
b85 }
733 void dfs(int u, int from_id) {
004   while(!adj[u].empty()) {
978     auto [v, id] = adj[u].back();
687     adj[u].pop_back();
c0a     if(seen_edges[id]) continue;
8a1     seen_edges[id] = true;
bd8     dfs(v, id);
adf   }
264   path.emplace_back(u);
d1c   if(from_id != -1) path_edges.emplace_back(from_id);
932 }
78b int get_start_node(int n) {
c1a   int st = 0;
19f   for(int u = 0; u < n; ++u) {
db8     if(deg[u] % 2) return u;
d70     if(deg[u] > 0) st = u;
1b8   }
aa0   return st;
fb9 }
ea5 bool get_eulerian_path(int n, int m) {
a9e   calc_deg(n);
f50   if(!has_eulerian_path(n)) return false;
b69   dfs(get_start_node(n), -1);
a31   return (int)path_edges.size() == m;
732 }
// remember to reverse path_edges!
```

1.8 Heavy Light Decomposition

```
//IS_EDGE: whether queries are on vertices or edges
//false: vertices, true: edges
cfd template<const bool IS_EDGE>
123 struct HeavyLightDecomposition {
6be   vector<int> tin, tout, sz, rin, p, nxt, h;
903   vector<vector<int>> adj;
8bd   int t;
812   HeavyLightDecomposition(int n) : tin(n), tout(n),
df7     sz(n), rin(n), p(n), nxt(n), h(n), adj(n) {}
58b   void add_edge(int u, int v) {
cc9     adj[u].push_back(v);
1ea     adj[v].push_back(u);
8ac   }
577   void set_root(int n) {
a34     t = 0;
5b4     p[n] = n;
581     h[n] = 0;
0fc     prep(n, n);
8c0     nxt[n] = n;
6d6     hld(n,n);
d2e   }
029   int get_lca(int u, int v) {
c60     while(!in_subtree(nxt[u], v)) u = p[nxt[u]];
c22     while(!in_subtree(nxt[v], u)) v = p[nxt[v]];
40a     return tin[u] < tin[v] ? u : v;
c52   }
6a9   bool in_subtree(int u, int v) {
        // is v tin the subtree of u
6d1     return tin[u] <= tin[v] && tin[v] < tout[u];
245   }
67a   template<typename T>
4be   void get_path_to_ancestor(int u, int anc, T&& get) {
        // returns ranges [l, r) that the path has
7ff     while(nxt[u] != nxt[anc]) {
9b7       get(tin[nxt[u]], tin[u] + 1);
a62       u = p[nxt[u]];
627     }
        // this includes the ancestor!
        // check if range [l,r) is valid when IS_EDGE
8f4     if(tin[anc] + IS_EDGE < tin[u] + 1) {
fc0       get(tin[anc] + IS_EDGE, tin[u] + 1);
529     }
cca   }
334   void prep(int u, int par) {
267     sz[u] = 1;
9a7     p[u] = par;
d44     for(int& v : adj[u]) {
9b9       if(v != par) {
294         h[v] = 1 + h[u];
f8f         prep(v, u);
cc3         sz[u] += sz[v];
083         if(sz[v] > sz[adj[u][0]] || adj[u][0] == par) {
072           swap(adj[u][0], v);
d4d         }
ea4       }
7a6     }
605   }
e77   void hld(int u, int par) {
2c6     tin[u] = t++;
22f     rin[tin[u]] = u;
d1c     for(auto v : adj[u]) {
d56       if(v == par) continue;
a02       nxt[v] = (v == adj[u][0] ? nxt[u] : v);
```

```
42c       hld(v, u);
d4e     }
5b9     tout[u] = t;
a6c   }
67f };
```

1.9 Min Cost Max Flow

```
d0a struct MinCostMaxFlow{
523   const Cost INF = numeric_limits<Cost>::max();
e9b   struct Edge {
df9     int to, next;
f23     Cap cap, flow;
cb9     Cost cost;
90c     Edge(int to, int next, Cap cap, Cost cost) : to(to), next(next),
    cap(cap), flow(0), cost(cost) {}
8fd     Cap res() const { return cap - flow; }
20a   };
05b   int m = 0, n;
321   vector<Edge> edges;
23b   vector<int> first;
ade   vector<Cap> neck;
35f   vector<Cost> dist, pot;
e3b   vector<int> from;
22d   vector<bool> inq;
26a   queue<int> q;
ce0   MinCostMaxFlow(int n) : n(n), first(n, -1), neck(n), pot(n) {}
780   void add_edge(int u, int v, Cap cap, Cost cost) {
df5     if(u != v) {
5c4       edges.emplace_back(v, first[u], cap, cost);
b6e       edges.emplace_back(u, first[v], 0, -cost);
d4c       first[u] = m++;
841       first[v] = m++;
fa8     }
a5e   }
cba   bool spfa(int s, int t) {
        //calculate initial potential, pot[u] = dist(s, u)
ef2     dist.assign(n, INF);
0b5     from.assign(n, -1);
350     inq.assign(n, false);
2de     neck[s] = numeric_limits<Cap>::max();
a93     dist[s] = 0;
08b     q.push(s);
14d     while(!q.empty()) {
352       auto u = q.front();
833       q.pop();
e0a       inq[u] = false;
d2e       for(int id = first[u]; id != -1; id = edges[id].next) {
f84         auto e = edges[id];
014         Cost w = e.cost + pot[u] - pot[e.to];
fe4         if(e.res() > 0 && dist[e.to] > dist[u] + w) {
6d0           from[e.to] = id;
5f5           dist[e.to] = dist[u] + w;
023           neck[e.to] = min(neck[u], e.res());
817           if(!inq[e.to]) {
b3a             inq[e.to] = true;
6f4             q.push(e.to);
7d7           }
cce         }
9b2       }
2a9     }
85d     return dist[t] < INF;
d12   }
9db   bool dijkstra(int s, int t) {
ef2     dist.assign(n, INF);
0b5     from.assign(n, -1);
2de     neck[s] = numeric_limits<Cap>::max();
c6f     using ii = pair<Cost, int>;
d9a     priority_queue<ii, vector<ii>, greater<ii>> pq;
6bd     pq.push({dist[s] = 0, s});
502     while(!pq.empty()) {
e18       auto [d_u, u] = pq.top();
716       pq.pop();
624       if(dist[u] != d_u) continue;
d2e       for(int id = first[u]; id != -1; id = edges[id].next) {
f84         auto e = edges[id];
014         Cost w = e.cost + pot[u] - pot[e.to];
fe4         if(e.res() > 0 && dist[e.to] > dist[u] + w) {
6d0           from[e.to] = id;
bee           pq.push({dist[e.to] = dist[u] + w, e.to});
023           neck[e.to] = min(neck[u], e.res());
e32         }
1f3       }
a68     }
85d     return dist[t] < INF;
5e8   }
1cb   pair<Cap, Cost> min_cost_max_flow(int s, int t, Cap k =
    numeric_limits<Cap>::max()) {
        // k : maximum flow allowed
717     Cap flow = 0;
247     Cost cost = 0;
        // in case of negative cost edges, use spfa + fix_pot
497     if(!spfa(s, t)) return {flow, cost};
f5d     fix_pot();
        // if graph is dense, change dijkstra to spfa
c28     while(flow < k && dijkstra(s, t)) {
e9d       Cap amt = min(neck[t], Cap(k - flow));
0d7       for(int v = t; v != s; v = edges[from[v] ^ 1].to) {
2ae         cost += edges[from[v]].cost * amt;
3b4         edges[from[v]].flow += amt;
60f         edges[from[v] ^ 1].flow -= amt;
48f       }
2e8       flow += amt;
f5d       fix_pot();
b0f     }
884     return {flow, cost};
aa4   }
2c0   void fix_pot() {
```

```
19f     for(int u = 0; u < n; ++u) {
35e       if(dist[u] < INF) {
ab7         pot[u] += dist[u];
bc9       }
ac5     }
011   }
437 };
```

1.10 Tarjan

```
e15 vector<int> tarjan(const vector<vector<int>>& adj) {
0b6   int n = (int)adj.size(), timer = 0, ncomps = 0;
551   enum State { unvisited, on_stack, visited };
5a2   vector<State> state(n, unvisited);
018   vector<int> low(n), tin(n), scc(n), stk;
3c1   auto dfs = [&](auto&& dfs, int u) -> void {
c09     low[u] = tin[u] = timer++;
967     stk.push_back(u);
3d2     state[u] = on_stack;
372     for(int v : adj[u]) {
5b0       if(state[v] == unvisited) {
d2a         dfs(dfs, v);
ab6         low[u] = min(low[u], low[v]);
fed       } else if(state[v] == on_stack) {
34f         low[u] = min(low[u], tin[v]);
013       }
3cd     }
b32     if(low[u] == tin[u]) {
d93       int v;
016       do {
97b         v = stk.back();
518         stk.pop_back();
143         state[v] = visited;
a95         scc[v] = ncomps;
ea2       } while(v != u);
c7e       ++ncomps;
39f     }
271   };
19f   for(int u = 0; u < n; ++u) {
7dd     if(state[u] == unvisited) {
22c       dfs(dfs, u);
c40     }
a07   }
9ab   return scc;
d7d }
```

1.11 Two Edge Component

```
10a struct TwoEdgeComponent {
551   enum State { unvisited, on_stack, visited };
34a   int n, timer, nedge;
63c   vector<vector<pair<int, int>>> adj;
b24   vector<State> state;
fb9   vector<int> tin, low, edge_stk;
00f   vector<bool> used_edge;
3a1   vector<vector<int>> two_edge_component;
f11   vector<pair<int, int>> bridge;
30a   TwoEdgeComponent(int n) : n(n), timer(0),
865   nedge(0), adj(n), state(n, unvisited),
d2d   tin(n), low(n) {}
010   void add_edge(int a, int b) {
394     adj[a].emplace_back(b, nedge);
a4b     adj[b].emplace_back(a, nedge);
ae8     nedge += 1;
151   }
845   void dfs(int u, int edge_id) {
c09     low[u] = tin[u] = timer++;
8e7     edge_stk.emplace_back(u);
3d2     state[u] = on_stack;
329     for(auto [v, id] : adj[u]) {
19b       if(edge_id == id) {
5e2         continue;
956       }
5b0       if(state[v] == unvisited) {
6a8         used_edge[id] = true;
bd8         dfs(v, id);
ab6         low[u] = min(low[u], low[v]);
975         if(low[v] > tin[u]) {
da3           bridge.emplace_back(u, v);
b08         }
469       } else if(state[v] == on_stack) {
34f         low[u] = min(low[u], tin[v]);
013       }
0e9     }
b32     if(low[u] == tin[u]) {
535       two_edge_component.emplace_back();
1ec       auto& comp = two_edge_component.back();
016       do {
54a         comp.emplace_back(edge_stk.back());
966         edge_stk.pop_back();
5f6         state[comp.back()] = visited;
c76       } while(comp.back() != u);
aa2     }
797   }
63d   void tarjan() {
bb9     used_edge.assign(nedge, false);
19f     for(int u = 0; u < n; ++u) {
7dd       if(state[u] == unvisited) {
787         dfs(u, -1);
10c       }
d6e     }
436   }
9a7   pair<vector<vector<int>>, vector<int>> build_bridge_tree() const {
124     int sz = (int)two_edge_component.size();
c8f     vector<vector<int>> g(sz);
```

```
201     vector<int> bcc(n);
cde     for(int id = 0; id < sz; ++id) {
c32       for(auto node : two_edge_component[id]) {
253         bcc[node] = id;
eba       }
ddd     }
19f     for(int u = 0; u < n; ++u) {
b40       for(auto [v, _] : adj[u]) {
1ae         if(bcc[u] != bcc[v]) {
571           g[bcc[u]].emplace_back(bcc[v]);
fb3         }
3be       }
2bd     }
79c     return {g, bcc};
4a7   }
454 };
```

2 Geometry

2.1 Convex Hull

```
67a template<typename T>
580 vector<Point<T>> convex_hull(vector<Point<T>> pts) {
cd7   sort(pts.begin(), pts.end());
b25   pts.erase(unique(begin(pts), end(pts)), end(pts));
d36   if(pts.size() <= 2) return pts;
de1   vector<Point<T>> upper(pts.size()), lower(pts.size());
ae3   int k = 0, l = 0;
e6e   for (auto p : pts) {
07e     while (k > 1 && !clockwise(upper[k - 1] - upper[k - 2], p - upper[k -
    1])) k -= 1;
1a4     while (l > 1 && !counterclockwise(lower[l - 1] - lower[l - 2], p -
    lower[l - 1])) l -= 1;
b8d     upper[k++] = lower[l++] = p;
0d6   }
858   upper.resize(k - 1), lower.resize(l);
4fe   lower.insert(lower.end(), upper.rbegin(), upper.rend() - 1);
b3e   return lower;
05a }

67a template<typename T>
2fb int maximize_dot_product(const vector<Point<T>>& h, const Point<T>& vec) {
      // might not work if there are 3 colinear points
10e   int n = (int)h.size();
1a4   int ans = 0;
a93   for(int rep = 0; rep < 2; ++rep) {
249     int lo = 0, hi = n - 1;
6e7     while(lo < hi) {
c86       int mid = (lo + hi) / 2;
77e       auto d1 = dot(h[mid + 1] - h[0], vec), d2 = dot(h[mid + 1] - h[mid],
    vec);
927       bool check = d2 > T(0);
1b2       if(rep == 0) check = check && d1 > T(0);
6b2       else check = check || d1 - d2 <= T(0);

afe       if(check) lo = mid + 1;
8c0       else hi = mid;
456     }
063     if(dot(h[ans], vec) < dot(h[lo], vec)) ans = lo;
90c   }
ba7   return ans;
d79 }
```

2.2 Halfplane Intersection

```
e14 using ld = long double; using DT = Double<ld>;
8be using PT = Point<DT>; using LI = Line<DT>;
7ba const DT INF = 1e18;
a16 vector<PT> halfplane_intersection(vector<LI> line) {
c39   vector<PT> box{PT(INF, INF), PT(-INF, INF), PT(-INF, -INF), PT(INF,
    -INF)};
8c3   for(int i = 0; i < 4; ++i) { // Add bounding box half-planes.
164     line.emplace_back(box[i], box[(i + 1) % 4] - box[i]);
ad9   }
      // Sort by angle and start algorithm
3bb   sort(begin(line), end(line), [&](LI u, LI v) {
e15     return polar_cmp(u.d, v.d);
1db   });
ce2   auto outside_halfplane = [&](LI hp, PT p) {
4da     return clockwise(hp.d, p - hp.A);
49d   };
c16   auto is_redundant = [&](LI a, LI b, LI c) {
c18     return outside_halfplane(a, line_intersection(b, c));
612   };
ae4   deque<LI> hp;
486   int len = 0;
542   for(int i = 0, n = (int)line.size(); i < n; ++i) {
        // Remove from the back of the deque while last half-plane is redundant
4ff     while(len > 1 && is_redundant(line[i], hp[len - 1], hp[len - 2])) {
3c1       hp.pop_back();
654       --len;
114     }
        // Remove from the front of the deque while first half-plane is
            redundant
e7a     while(len > 1 && is_redundant(line[i], hp[0], hp[1])) {
5a0       hp.pop_front();
654       --len;
f20     }
        // Special case check: Parallel half-planes
d8a     if(len > 0 && cross(line[i].d, hp[len - 1].d) == DT(0)) {
          // Opposite parallel half-planes that ended up checked against each
              other.
464       if(dot(line[i].d, hp[len - 1].d) < DT(0)) {
aed         return vector<PT>();
```

```
830       }
          // Same direction half-plane: keep only the leftmost half-plane.
a7a       if(outside_halfplane(line[i], hp[len - 1].A)) {
3c1         hp.pop_back();
654         --len;
03d       } else continue;
134     }
          // Add new half-plane
20a     hp.push_back(line[i]);
250     ++len;
ed9   }
      // Final cleanup: Check half-planes at the front against the back and
          vice-versa
68c   while(len > 2 && is_redundant(hp[0], hp[len - 1], hp[len - 2])) {
3c1     hp.pop_back();
654     --len;
06c   }
51d   while(len > 2 && is_redundant(hp[len - 1], hp[0], hp[1])) {
5a0     hp.pop_front();
654     --len;
dc2   }
      // Report empty intersection if necessary
3b4   if (len < 3) {
aed     return vector<PT>();
df9   }
      // Reconstruct the convex polygon from the remaining half-planes.
228   vector<PT> inter(len);
a9e   for(int i = 0; i < len; ++i) {
f8d     int j = i + 1 == len ? 0 : i + 1;
2f9     inter[i] = line_intersection(hp[i], hp[j]);
741   }
c17   return inter;
c96 }
```

2.3 Minkowski Sum

```
// Given two convex polygons, calculate the convex polygon represented by
    their sum
// a_i in poly A, b_j in poly B then a_i + b_j in poly A+B
67a template<typename T>
dfc void reorder_polygon(vector<Point<T>>& P){
65a   size_t pos = 0;
81f   for(size_t i = 1; i < P.size(); i++){
b8d     if(P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x < P[pos].x)) {
e4c       pos = i;
1f6     }
ffd   }
98e   rotate(P.begin(), P.begin() + pos, P.end());
ab8 }
// points ordered ccw
67a template<typename T>
e3f vector<Point<T>> minkowski(vector<Point<T>> P, vector<Point<T>> Q){
      // the first vertex must be the lowest
159   reorder_polygon(P);
fad   reorder_polygon(Q);
      // we must ensure cyclic indexing
642   P.push_back(P[0]);
6ed   P.push_back(P[1]);
406   Q.push_back(Q[0]);
d11   Q.push_back(Q[1]);
912   vector<Point<T>> result;
829   for(size_t i = 0, j = 0; i < P.size() - 2 || j < Q.size() - 2; ){
d81     result.push_back(P[i] + Q[j]);
b60     auto c = cross(P[i + 1] - P[i], Q[j + 1] - Q[j]);
2ff     if(c >= T(0) && i < P.size() - 2) {
c7e       ++i;
f21     }
689     if(c <= T(0) && j < Q.size() - 2) {
3cc       ++j;
ffb     }
15d   }
dc8   return result;
088 }
```

2.4 Point 2D

```
67a template<typename T>
f26 struct Point {
645   T x, y;
0fa   Point(T x = 0, T y = 0) : x(x), y(y) {}
056   Point operator+(const Point& rhs) const { return Point(x + rhs.x, y +
    rhs.y); }
be4   Point operator-(const Point& rhs) const { return Point(x - rhs.x, y -
    rhs.y); }
96f   Point operator-() const { return Point() - *this; }
2f9   Point operator*(T c) const { return Point(x * c, y * c); }
e4d   Point operator/(T c) const { return Point(x / c, y / c); }
175   bool operator<(const Point& rhs) const {
a9b     if(x == rhs.x) return y < rhs.y;
cf3     return x < rhs.x;
f03   }
ef9   bool operator==(const Point& rhs) const { return x == rhs.x && y ==
    rhs.y; }
14c   bool operator!=(const Point& rhs) const { return !(*this == rhs); }
398   template<typename F>
56d   explicit operator Point<F>() const { return Point<F>(F(x), F(y)); }
efa   friend ostream& operator<<(ostream& os, const Point& o) {
37d     return os << o.x << ' ' << o.y;
295   }
061   friend istream& operator>>(istream& is, Point& o) {
b56     return is >> o.x >> o.y;
cdf   }
c96 };
67a template<typename T>
1d2 T dot(Point<T> u, Point<T> v) { return u.x * v.x + u.y * v.y; }
67a template<typename T>
```

```
b84 T cross(Point<T> u, Point<T> v) { return u.x * v.y - u.y * v.x; }
e5a template<typename T = double>
d6c T norm(Point<T> u) { return sqrt(dot(u, u)); }
e5a template<typename T = double>
e3a Point<T> proj(Point<T> u, Point<T> v) { return v * (dot(u, v) / dot(v,
    v)); }
67a template<typename T>
fc4 bool counterclockwise(Point<T> u, Point<T> v) { return cross(u, v) > T(0);
    }
67a template<typename T>
27c bool clockwise(Point<T> u, Point<T> v) { return cross(u, v) < T(0); }
67a template<typename T>
b92 Point<T> rotateCCW90(Point<T> u) { return Point<T>(-u.y, u.x); }
67a template<typename T>
dd9 Point<T> rotateCW90(Point<T> u) { return Point<T>(u.y, -u.x); }
e5a template<typename T = double>
568 Point<T> rotateCCW(Point<T> u, T t) {
695   return Point<T>(u.x * cos(t) - u.y * sin(t), u.x * sin(t) + u.y *
    cos(t));
4a8 }
e5a template<typename T = double>
cbc T angle(Point<T> u, Point<T> v) { return acos(dot(u, v) / (norm(u) *
    norm(v))); }
```

2.5 Point 3D

```
67a template<typename T>
f26 struct Point {
329   T x, y, z;
62f   Point(T x = 0, T y = 0, T z = 0) : x(x), y(y), z(z) {}
c7d   Point operator+(const Point& o) const { return Point(x + o.x, y + o.y, z
    + o.z);}
c4c   Point operator-(const Point& o) const { return Point(x - o.x, y - o.y, z
    - o.z);}
602   friend ostream& operator<<(ostream& os, const Point& o) { return os <<
    o.x << ' ' << o.y << ' ' << o.z; }
a07   friend istream& operator>>(istream& is, Point& o) { return is >> o.x >>
    o.y >> o.z; }
398   template<typename F>
102   operator Point<F>() const {
4d6     return Point<F>(F(x), F(y), F(z));
1ef   }
135 };
67a template<typename T>
455 T dot(Point<T> u, Point<T> v) { return u.x * v.x + u.y * v.y + u.z * v.z; }
67a template<typename T>
302 Point<T> cross(Point<T> u, Point<T> v) {
45c   return Point(u.y * v.z - u.z * v.y, -u.x * v.z + v.x * u.z, u.x * v.y -
    v.x * u.y);
613 }
// returns if P is on the plane determined by vectors u and v
be6 bool point_on_plane(Point<__int128_t> u, Point<__int128_t> v,
    Point<__int128_t> P) {
914   auto w = cross(u, v);
1e2   return dot(w, P) == 0;
e0f }
1c7 bool same_side(Point<__int128_t> a, Point<__int128_t> b, Point<__int128_t>
    c, Point<__int128_t> P) {
31c   auto u = cross(b - a, c - a);
e6f   auto v = cross(b - a, P - a);
cb1   return dot(u, v) >= 0;
d3b }
674 bool point_inside_triangle(Point<int> a, Point<int> b, Point<int> c,
    Point<int> P) {
b14   if(!point_on_plane(b - a, c - a, P - a)) {
d1f     return false;
7c9   }
1dc   return same_side(a, b, c, P) && same_side(b, c, a, P) && same_side(c, a,
    b, P);
74d }
```

2.6 Polar ordering

```
67a template<typename T>
90b bool is_up(Point<T> u) {
e2d   if(u.y > T(0)) return true;
620   return u.y == T(0) && u.x >= T(0);
b4b }
67a template<typename T>
aa5 bool polar_cmp(Point<T> u, Point<T> v) {
ef2   if(is_up(u) == is_up(v)) return counterclockwise(u, v);
9fd   return is_up(u) > is_up(v);
c21 }
67a template<typename T>
6f8 bool same_half_plane(Point<T> u, Point<T> v) {
aab   if(cross(u, v) > T(0)) return true;
bd9   return cross(u, v) == T(0) && dot(u, v) >= T(0);
ca6 }
```

2.7 Polygon

```
67a template<typename T>
634 bool same_side(Point<T> P, Point<T> A, Point<T> B, Point<T> C) {
3e4   T u = cross(B - A, P - A);
000   T v = cross(B - A, C - A);
b73   int x = u == T(0) ? 0 : u < T(0) ? -1 : +1;
86a   int y = v == T(0) ? 0 : v < T(0) ? -1 : +1;
1f3   return x * y >= 0;
2c8 }
67a template<typename T>
fea bool point_inside_triangle(Point<T> P, Point<T> A, Point<T> B, Point<T> C)
    {
```

```
cbc   return same_side(P, A, B, C) && same_side(P, B, C, A) && same_side(P, C,
    A, B);
350 }
// polygon must be ordered counterclockwise
67a template<typename T>
77f bool point_inside_convex_polygon(Point<T> P, const vector<Point<T>>& poly)
    {
7e9   if(poly.size() == 1) return P == poly[0];
982   if(poly.size() == 2) return on_segment(P, Line(poly[0], poly[1] -
    poly[0]));
961   int l = 1, r = (int)poly.size() - 1;
219   while(r - l > 1) {
ee4     int m = (l + r) / 2;
f21     if(clockwise(poly[m] - poly[0], P - poly[0])) {
3e2       r = m;
c7d     } else {
8a6       l = m;
903     }
b67   }
ac7   return point_inside_triangle(P, poly[0], poly[l], poly[l + 1]);
603 }
67a template<typename T>
260 double polygon_area(const vector<Point<T>>& poly) {
f13   double area = 0;
f6a   for(int i = 0, n = (int)poly.size(); i < n; ++i) {
a91     int j = i + 1 == n ? 0 : i + 1;
140     area += cross(poly[i], poly[j]);
a87   }
d3f   return abs(area) / 2;
3e1 }
```

2.8 Primitive Intersections

```
// Line: A + t * d
// Segment: [A, B], B = A + 1 * d
// Beware degenerate cases: d = 0!
67a template<typename T>
72c struct Line {
13f   Point<T> A, d;
9ca   Line(Point<T> A = Point<T>(), Point<T> d = Point<T>()) : A(A), d(d) {}
811   Point<T> B() const { return A + d; }
398   template<typename F>
b6f   explicit operator Line<F>() const { return Line<F>(Point<F>(A),
    Point<F>(d)); };
256 };
67a template<typename T>
d5e bool on_line(Point<T> P, Line<T> line) { return cross(P - line.A, line.d)
    == T(0); }
67a template<typename T>
a78 bool on_segment(Point<T> P, Line<T> seg) {  return on_line(P, seg) &&
    dot(seg.A - P, seg.B() - P) <= T(0); }
67a template<typename T>
935 bool on_ray(Point<T> P, Line<T> ray) { return on_line(P, ray) && dot(P -
    ray.A, ray.d) >= T(0); }
e5a template<typename T = double>
398 T point_line_distance(Point<T> P, Line<T> line) { return abs(cross(line.d,
    P - line.A)) / norm(line.d); }
e5a template<typename T = double>
936 T point_segment_distance(Point<T> P, Line<T> seg) {
074   if(dot(seg.d, P - seg.A) < T(0)) return norm(P - seg.A);
32d   if(dot(P - seg.B(), -seg.d) < T(0)) return norm(P - seg.B());
f86   return point_line_distance(P, seg);
764 }
e5a template<typename T = double>
67c T point_ray_distance(Point<T> P, Line<T> seg) {
074   if(dot(seg.d, P - seg.A) < T(0)) return norm(P - seg.A);
f86   return point_line_distance(P, seg);
46b }
e5a template<typename T = double>
e3f Point<T> line_projection(Point<T> P, Line<T> line) { return line.A +
    proj(P - line.A, line.d); }
67a template<typename T>
d1e bool collinear(Line<T> line1, Line<T> line2) { return cross(line1.d,
    line2.d) == T(0); }
67a template<typename T>
d88 bool same_line(Line<T> line1, Line<T> line2) { return collinear(line1,
    line2) && cross(line1.A - line2.A, line1.d) == T(0); }
e5a template<typename T = double>
1ed T intersection_time(Line<T> line1, Line<T> line2) { return cross(line2.A -
    line1.A, line2.d) / cross(line1.d, line2.d); }
e5a template<typename T = double>
9d8 Point<T> line_intersection(Line<T> line1, Line<T> line2) { return line1.A
    + line1.d * intersection_time(line1, line2); }
cb1 template<typename T = Double<double>>
fac vector<Point<T>> segment_segment_intersection(Line<T> seg1, Line<T> seg2) {
53f   vector<Point<T>> intersection;
ab9   auto dd = cross(seg1.d, seg2.d);
b00   auto ls = cross(seg2.A - seg1.A, seg1.d);
d39   if(dd == T(0) && ls == T(0)) {
622     if(dot(seg1.d, seg2.d) < T(0)) {
106       seg2 = Line(seg2.B(), seg2.A - seg2.B());
4b7     }
08a     Point<T> L = dot(seg2.A - seg1.A, seg1.d) < T(0) ? seg1.A : seg2.A;
cab     Point<T> R = dot(seg2.B() - seg1.B(), seg1.d) < T(0) ? seg2.B() :
    seg1.B();
8b0     if(dot(R - L, seg1.d) >= T(0)) {
da8       intersection.emplace_back(L);
0cf       if(L != R) intersection.emplace_back(R);
6a8     }
488   } else if(dd != T(0)) {
ab1     auto rs = cross(seg2.A - seg1.A, seg2.d);
93a     if(dd < T(0)) dd = -dd, ls = -ls, rs = -rs;
479     bool intersect = 0 <= ls && ls <= dd && 0 <= rs && rs <= dd;
0fb     if(intersect) {
cdd       intersection.emplace_back(seg1.A + seg1.d * rs / dd);
e99     }
45e   }
```

```
74d   return intersection;
a84 }
cb1 template<typename T = Double<double>>
2ba vector<Point<T>> circle_line_intersection(Point<T> C, T r, Line<T> line) {
83b   vector<Point<T>> intersections;
822   Point<T> P = line_projection(C, line);
087   T h = norm(P - C);
fe4   if(h == r) {
359     intersections.emplace_back(P);
b90   } else if(h < r) {
031     T x = sqrt(r * r - h * h);
d16     line.d = line.d / norm(line.d);
fce     for(T d : {-1, +1}) {
b09       intersections.emplace_back(P + line.d * (d * x));
979     }
2b0   }
d99   return intersections;
957 }
cb1 template<typename T = Double<double>>
753 vector<Point<T>> circle_circle_intersection(Point<T> C1, T r1, Point<T>
    C2, T r2) {
7fa   if(C1 == C2) return {};
c9b   T a = 2 * (C1.x - C2.x);
9f8   T b = 2 * (C1.y - C2.y);
daf   T c = (dot(C2, C2) - r2 * r2) - (dot(C1, C1) - r1 * r1);
b0c   Line<T> line;
3e3   if(a == T(0)) {
a05     line = Line(Point<T>(0, -c / b), Point<T>(1, 0));
8a5   } else if(b == T(0)) {
1b3     line = Line(Point<T>(-c / a, 0), Point<T>(0, 1));
6e7   } else {
11b     line = Line(Point<T>(0, -c / b), Point<T>(b, -a));
835   }
388   return circle_line_intersection(C1, r1, line);
df2 }
cb1 template<typename T = Double<double>>
2e0 vector<Point<T>> circle_point_tangent(Point<T> C, T r, Point<T> P) {
f1c   vector<Point<T>> tg;
5e7   T d = norm(C - P);
c40   T xx = dot(C - P, C - P) - r * r;
fb9   if(xx == T(0)) {
4ea     tg.emplace_back(P);
b45   } else if(xx > T(0)) {
d8f     T x = sqrt(xx);
50f     Point<T> u = (C - P) * (x / d);
ee6     Point<T> A = P + rotateCCW<T>(u, acos(x / d));
384     Point<T> B = P + rotateCCW<T>(u, -acos(x / d));
25e     tg.emplace_back(A);
10a     tg.emplace_back(B);
825   }
a1d   return tg;
9ef }
```

3 string

3.1 Aho-Corasick

```
123 struct AhoType {
847   static const int ALPHA = 26;
f03   static int f(char c) { return c - 'a'; }
e07 };
29b template<typename AhoType>
51f struct AhoCorasick {
bf2   struct Node {
64c     int nxt[AhoType::ALPHA] {};
0d9     int p = 0, ch = 0, len = 0;
e7a     int link = 0;
79f     int occ_link = 0;
f4f     Node(int p = 0, int ch = 0, int len = 0) : p(p), ch(ch), len(len) {}
8ee   };
8ed   vector<Node> tr;
69b   AhoCorasick() : tr(1) {}
1f7   template<typename Iterator>
2ca   void add_word(Iterator first, Iterator last) {
ac3     int cur = 0, len = 1;
68c     for(; first != last; ++first) {
ed9       auto ch = AhoType::f(*first);
4f3       if(tr[cur].nxt[ch] == 0) {
9bf         tr[cur].nxt[ch] = int(tr.size());
6cc         tr.emplace_back(cur, ch, len);
b7c       }
bee       cur = tr[cur].nxt[ch];
250       ++len;
159     }
d91     tr[cur].occ_link = cur;
fa3   }
0a8   void build() {
a36     vector<int> bfs(int(tr.size()));
2aa     int s = 0, t = 1;
d33     while(s < t) {
b21       int v = bfs[s++], u = tr[v].link;
f9e       if(tr[v].occ_link == 0) {
99b         tr[v].occ_link = tr[u].occ_link;
e75       }
609       for(int ch = 0; ch < AhoType::ALPHA; ++ch) {
31d         auto& nxt = tr[v].nxt[ch];
9fa         if(nxt == 0) {
2ca           nxt = tr[u].nxt[ch];
95c         } else {
fe1           tr[nxt].link = v > 0 ? tr[u].nxt[ch] : 0;
47d           bfs[t++] = nxt;
fad         }
d85       }
fbe     }
7ff   }
a74   template<typename Iterator, typename Report>
2ee   void get_all_matches(Iterator first, Iterator last, Report&& report)
```

```
    const {
e09     for(int cur = 0, i = 0; first != last; ++i, ++first) {
ed9       auto ch = AhoType::f(*first);
bee       cur = tr[cur].nxt[ch];
f2c       for(int v = tr[cur].occ_link; v > 0; v = tr[tr[v].link].occ_link) {
881         report(i, v);
dde       }
5b7     }
d90   }
67a   template<typename T>
578   int get_next(int cur, T ch) const { return tr[cur].nxt[AhoType::f(ch)]; }
a0a };
```

3.2 KMP

```
67a template<typename T>
8fc vector<int> get_border(const T& s) {
c73   int n = (int)s.size();
d84   vector<int> border(n);
677   for(int i = 1, j = 0; i < n; ++i) {
45d     while(j > 0 && s[i] != s[j]) {
3ce       j = border[j - 1];
60a     }
aec     if(s[i] == s[j]) {
3cc       ++j;
43f     }
805     border[i] = j;
09a   }
887   return border;
ee5 }
2a1 template<typename T, typename F>
819 void match_pattern(const T& txt, const T& pat, const vector<int>& border,
    F get) {
860   int n = (int)txt.size();
f8e   int m = (int)pat.size();
cb2   for(int i = 0, j = 0; i < n; ++i) {
b0c     while(j > 0 && txt[i] != pat[j]) {
3ce       j = border[j - 1];
2e8     }
1fc     if(txt[i] == pat[j]) {
3cc       ++j;
5c5     }
c11     if(j == m) {
ca1       get(i - m + 1);
3ce       j = border[j - 1];
3e4     }
b5c   }
e06 }
```

3.3 Rabin Karp

```
29b template<typename Mint>
e3e struct RabinKarp {
1a8   int n;
84c   vector<Mint> p, pw;
464   RabinKarp() {}
67a   template<typename T>
51b   RabinKarp(const T& s, Mint C) : n(int(s.size())) {
a46     pw.assign(n + 1, 1);
961     p.assign(n + 1, 0);
3f2     for(int i = 1; i <= n; ++i) {
b3c       pw[i] = pw[i - 1] * C;
c9b       p[i] = p[i - 1] * C + s[i - 1];
a58     }
e73   }
813   Mint hash(int i, int len) const {
db1     return (p[i + len] - pw[len] * p[i]);
bea   }
314 };
29b template<typename Mint>
61c struct Hash {
1a8   int n;
0c3   RabinKarp<Mint> rab[2], rev_rab[2];
67a   template<typename T>
635   Hash(const T& s, Mint C0 = 727, Mint C1 = 137) : n((int)s.size()) {
c17     Mint C[2] = {C0, C1};
256     auto rev_s = s;
47f     reverse(begin(rev_s), end(rev_s));
28d     for(int e = 0; e < 2; ++e) {
440       rab[e] = RabinKarp<Mint>(s, C[e]);
ed0       rev_rab[e] = RabinKarp<Mint>(rev_s, C[e]);
78f     }
84f   }
49f   pair<Mint, Mint> get_hash(int l, int r) const {
b7c     return {rab[0].hash(l, r - l), rab[1].hash(l, r - l)};
1d4   }
c37   pair<Mint, Mint> get_reverse_hash(int l, int r) const {
e88     return {rev_rab[0].hash(n - r, r - l), rev_rab[1].hash(n - r, r - l)};
e64   }
cc1   bool is_palindrome(int l, int r) const {
1ce     return get_hash(l, r) == get_reverse_hash(l, r);
cec   }
fdc };
```

3.4 Suffix Array

```
5a4 void count_sort(vector<int>& sa, const vector<int>& c) {
609   int n = (int)sa.size();
007   vector<int> cnt(n + 1), sa_new(n);
876   for(int x : c) {
0ea     ++cnt[x + 1];
9ef   }
6fa   for(int i = 1; i < n; ++i) {
657     cnt[i] += cnt[i - 1];
```

```
384   }
02c   for(int x : sa) {
3df     sa_new[cnt[c[x]]++] = x;
cb8   }
bd6   sa.swap(sa_new);
1de }
67a template<typename T>
5ed vector<int> suffix_array(const T& s) {
c73   int n = (int)s.size();
1ad   auto mod = [&n](int x) {
d0f     return x < 0 ? x + n : x >= n ? x - n : x;
957   };
7a9   vector<int> sa(n), c(n);
67b   iota(begin(sa), end(sa), 0);
e37   sort(begin(sa), end(sa), [&](int a, int b) {
c40     return s[a] < s[b];
f90   });
cbe   int m = 0;
9d1   c[sa[0]] = m++;
6fa   for(int i = 1; i < n; ++i) {
30e     c[sa[i]] = s[sa[i]] != s[sa[i - 1]] ? m++ : m - 1;
f78   }
a16   for(int h = 1; h < n && m < n; h <<= 1) {
607     for(int& x : sa) {
154       x = mod(x - h);
87b     }
246     count_sort(sa, c);
0f4     vector<int> c_new(n);
31a     m = 0;
6c8     c_new[sa[0]] = m++;
6fa     for(int i = 1; i < n; ++i) {
691       pair<int,int> prev = {c[sa[i - 1]], c[mod(sa[i - 1] + h)]};
158       pair<int,int> cur = {c[sa[i]],c[mod(sa[i] + h)]};
58a       c_new[sa[i]] = prev != cur ? m++ : m - 1;
c9f     }
a97     c.swap(c_new);
517   }
db7   return sa;
b14 }
//lcp[0] = 0
//lcp[i] = longest common prefix(sa[i - 1], sa[i])
67a template<typename T>
a95 vector<int> get_lcp(const T& s, const vector<int>& sa) {
c73   int n = (int)s.size();
36f   vector<int> lcp(n), inv(n);
163   for(int i = 0; i < n; ++i) inv[sa[i]] = i;
3f2   for(int i = 0, k = 0; i < n - 1; ++i, k = k > 0 ? k - 1 : 0) {
598     int j = sa[inv[i] - 1];
d2d     while(s[i + k] == s[j + k]) {
caa       ++k;
238     }
763     lcp[inv[i]] = k;
565   }
5ed   return lcp;
b7a }
```

4 Miscellaneous

4.1 Closest Pair - DNC

```
67a template<typename T>
24c long long sq(T a) { return 1ll * a * a; }
67a template<typename T>
675 long long dist(pair<T, T> a, pair<T, T> b) {
090   return sq(a.first - b.first) + sq(a.second - b.second);
7ca }
67a template<typename T>
37c long long divide_and_conquer(const vector<pair<T, T>>& px, const
    vector<pair<T, T>>& py) {
7ad   int n = (int)px.size();
505   auto min_distance = numeric_limits<long long>::max();
e3b   if(n == 1) {
dd4     return min_distance;
9eb   }
5fe   auto lx = vector(begin(px), begin(px) + n / 2);
904   auto rx = vector(begin(px) + n / 2, end(px));
2b8   vector<pair<T, T>> ly, ry;
d7f   auto pivot = px[n / 2 - 1];
369   for(auto p : py) {
4db     if(p < pivot) {
de0       ly.emplace_back(p);
e44     } else {
8cf       ry.emplace_back(p);
f6c     }
d3d   }
409   auto ld = divide_and_conquer(lx, ly);
389   auto rd = divide_and_conquer(rx, ry);
26c   min_distance = min(ld, rd);
e73   vector<pair<T, T>> stripe;
369   for(auto p : py) {
6a3     if(sq(p.first - pivot.first) < min_distance) {
828       stripe.emplace_back(p);
c22     }
c7b   }
a2e   for(int i = 0, len = (int)stripe.size(); i < len; ++i) {
373     for(int j = i + 1; j < len && sq(stripe[i].second - stripe[j].second)
    < min_distance; ++j) {
1b9       min_distance = min(min_distance, dist(stripe[i], stripe[j]));
d8a     }
874   }
dd4   return min_distance;
abd }
67a template<typename T>
77e long long closest_pair(vector<pair<T, T>> px) {
3af   auto py = px;
978   sort(begin(px), end(px));
1de   sort(begin(py), end(py), [](auto a, auto b) {
293     return tie(a.second, a.first) < tie(b.second, b.first);
d4b   });
892   return divide_and_conquer(px, py);
```

```
3ce }
```

4.2 Color Update

```
28f template<typename T, typename Color>
f1d struct ColorUpdate {
3d4   struct Range {
dbb     T l, r;
d8d     Color v;
1cb     Range(T l) : l(l) {}
297     Range(T l, T r, Color v) : l(l), r(r), v(v) {}
a28     bool operator<(const Range& rhs) const { return l < rhs.l; }
c2d   };
4a2   set<Range> ranges;
398   template<typename F>
35f   Range update(T l, T r, Color v, F&& get) {
efb     auto it = ranges.lower_bound(l);
a43     if(it != ranges.begin()) {
b32       if(prev(it)->r > l) {
fc2         --it;
bf0         auto cur = *it;
ec6         it = ranges.erase(it);
2c6         it = ranges.emplace_hint(it, cur.l, l, cur.v);
5b4         it = ranges.emplace_hint(it, l, cur.r, cur.v);
5b2       }
5af     }
fa2     for(; it != ranges.end() && it->r <= r;) {
bf0       auto cur = *it;
ec6       it = ranges.erase(it);
aa0       get(cur.l, cur.r, cur.v);
7c9     }
67a     if(it != ranges.end()) {
b4b       if(it->l < r) {
bf0         auto cur = *it;
ec6         it = ranges.erase(it);
234         get(cur.l, r, cur.v);
ba9         it = ranges.emplace_hint(it, r, cur.r, cur.v);
9bd       }
098     }
699     it = ranges.emplace_hint(it, l, r, v);
768     return Range(l, r, v);
1b4   }
806 };
```

4.3 Coordinate Compression

```
67a template<typename T>
851 struct CoordinateCompression {
517   vector<T> v;
6a3   void push(const T& a) { v.push_back(a); }
a05   int build() {
484     sort(begin(v), end(v));
c0c     v.erase(unique(begin(v), end(v)), end(v));
f0c     return (int)v.size();
518   }
5b6   int operator[](const T& a) const {
50a     auto it = lower_bound(begin(v), end(v), a);
154     return int(it - begin(v));
52e   }
4d7 };
```

4.4 Custom Double

```
bf6 const double EPS = 1e-9;
e5a template<typename T = double>
3af int sign(T x) { return abs(x) < EPS ? 0 : x < 0 ? -1 : +1; }
e5a template<typename T = double>
b58 struct Double {
bad   T x;
99f   Double(T x = 0) : x(x) {}
7b6   bool operator==(Double rhs) const { return sign(x - rhs.x) == 0; }
3ce   bool operator!=(Double rhs) const { return sign(x - rhs.x) != 0; }
b3e   bool operator<(Double rhs)  const { return sign(x - rhs.x)  < 0; }
2d2   bool operator<=(Double rhs) const { return sign(x - rhs.x) <= 0; }
0c9   bool operator>(Double rhs)  const { return sign(x - rhs.x)  > 0; }
624   bool operator>=(Double rhs) const { return sign(x - rhs.x) >= 0; }
7fa   friend ostream& operator<<(ostream& os, const Double& o) { return os <<
    o.x; }
963   friend istream& operator>>(istream& is, Double& o) { return is >> o.x; }
861   operator T() const { return x; } // implicit conversion
281 };
cc2 using DT = Double<long double>;
// make sure comparisons are always between same type
// avoid problems with implicit conversion
```

4.5 Golden Ratio

```
// use for speed up ternary searches
67e const dt gr = (sqrt(5) + 1) / 2, EPS = 1e-7;
398 template<typename F>
168 dt golden_ratio_search(dt lo, dt hi, F&& f) {
663   dt x1 = hi - (gr - 1) * (hi - lo), x2 = lo + (gr - 1) * (hi - lo);
07c   dt f1 = f(x1), f2 = f(x2);
bbd   for(; hi - lo > EPS;) {
676     if(f1 > f2) {
827       hi = x2; x2 = x1; f2 = f1;
25f       x1 = hi - (gr - 1) * (hi - lo);
503       f1 = f(x1);
07a     } else {
aee       lo = x1; x1 = x2; f1 = f2;
```

```
fca       x2 = lo + (gr - 1) * (hi - lo);
862       f2 = f(x2);
47e     }
54c   }
cb9   return x1;
5f9 }
```

4.6 MO

```
327 vector<int> mosort(const vector<pair<int, int>>& query, const int B) {
d60   int q = (int)query.size();
b89   vector<pair<int ,int>> query_id(q);
226   for(int i = 0; i < q; ++i) {
4db     auto [l, r] = query[i];
05d     auto x = l / B;
3fe     auto y = x & 1 ? -r : +r;
803     query_id[i] = {x, y};
a2c   }
f39   vector<int> ord(q);
053   iota(begin(ord), end(ord), 0);
430   sort(begin(ord), end(ord), [&](int i, int j) {
3f0     return query_id[i] < query_id[j];
0e1   });
342   return ord;
6c1 }
288 int64_t xy2d_hilbert(int n, int x, int y) {
9a1   int64_t d = 0;
438   for (int s = n / 2; s > 0; s >>= 1) {
1ba     int rx = (x & s) > 0;
c74     int ry = (y & s) > 0;
03e     d += (int64_t) s * s * ((3 * ry) ^ rx);
e3f     if (rx == 0) {
21b       if (ry == 1) {
731         x = s - 1 - x;
d41         y = s - 1 - y;
d95       }
9dd       swap(x, y);
522     }
dfa   }
be2   return d;
07d }
015 vector<int> mosort_hilbert(int n, const vector<pair<int, int>>& query) {
553   int k = n > 1 ? __lg(n - 1) + 1 : 0;
b03   n = 1 << k;
d60   int q = (int)query.size();
615   vector<int64_t> id(q);
abf   for (int i = 0; i < q; i++) {
4db     auto [l, r] = query[i];
bbb     id[i] = xy2d_hilbert(n, l, r);
b91   }
f39   vector<int> ord(q);
053   iota(begin(ord), end(ord), 0);
430   sort(begin(ord), end(ord), [&](int i, int j) {
a4e     return id[i] < id[j];
d4f   });
342   return ord;
7c8 }
2a1 template<typename T, typename F>
d46 vector<F> process_query(const vector<T> a, const vector<pair<int, int>>&
    query) {
8ec   int n = (int)a.size();
d60   int q = (int)query.size();
9ce   auto ord = mosort_hilbert(n, query);
      //auto ord = mosort(query, sqrt(n) + 1);
22a   vector<F> ans(q);
195   int mo_l = 0, mo_r = 0;
0a5   F mo_ans{};
721   auto add = [&](T x) {
        /* */
e07   };
59e   auto remove = [&](T x) {
        /* */
adb   };
8b0   for(int i : ord) {
4db     auto [l, r] = query[i];
e98     while(mo_l > l) add(a[--mo_l]);
a23     while(mo_r < r) add(a[mo_r++]);
09d     while(mo_l < l) remove(a[mo_l++]);
c58     while(mo_r > r) remove(a[--mo_r]);
19e     ans[i] = mo_ans;
7e5   }
ba7   return ans;
1f3 }
```

4.7 Parallel Binary Search

```
a46 vector<int> L(n, 0), R(n, q);
e99 for(int l = 0; l < 20; ++l) {
7cb   vector<vector<int>> on(q);
bae   for(int i = 0; i < n; ++i) {
fcf     if(L[i] == R[i]) continue;
f0a     int m = (L[i] + R[i])/ 2;
228     on[m].emplace_back(i);
c98   }
      // initialize some structure
4f0   auto add = [&](int i) { /* add i-th element to the data structure */ };
694   auto check = [&](int i) { /* check condition for current prefix of
    elements to the i-th query */ };
3d2   for(int m = 0; m < q; ++m) {
d9a     add(m); // maintain prefix of elements
410     for(auto i : on[m]) {
ec0       if(check(i)) R[i] = m;
c71       else L[i] = m + 1;
2b0     }
980   }
f33 }
```

4.8 Tree Hash

```
cdc map<vector<int>, int> hasher;
c52 int hashify(vector<int> x) {
60f   sort(begin(x), end(x));
c78   if(!hasher[x]) {
93b     hasher[x] = (int)hasher.size();
da6   }
464   return hasher[x];
93c }
5fe int get_hash(int u, int p) { // get a "hash" of v's subtree
46d   vector<int> children;
4d5   for(int v: g[u]) {
40d     if(v == p) {
7aa       continue
2f2     }
343     children.push_back(get_hash(v, u));
c7d   }
8f9   return hashify(children);
ab0 }
```

5 Data Structures

5.1 Fenwick Tree

```
67a template<typename T>
0f1 struct FenwickTree {
79d   static int lsb(int b) { return b & -b; }
1a8   int n;
1d9   vector<T> ft;
b5d   FenwickTree(int n = 0) : n(n), ft(n + 1, T()) {}
1f7   template<typename Iterator>
381   FenwickTree(Iterator first, Iterator last) : FenwickTree(int(last -
    first)) {
bae     for (int i = 0; i < n; ++i) {
609       ft[i + 1] = first[i] + ft[i];
480     }
d21     for (int i = n; i >= 1; --i) {
78d       ft[i] -= ft[i - lsb(i)];
71a     }
94b   }
3b9   void update(int x, const T& val) {
c71     for(++x; x <= n; x += lsb(x)) {
f28       ft[x] += val;
090     }
19d   }
612   T query(int x) const { //query on [0,x)
d8e     T ret{};
ffd     for(; x > 0; x -= lsb(x)) {
88c       ret += ft[x];
89f     }
edf     return ret;
618   }
6a1   T query(int l, int r) const { // query on [l,r)
3b4     if(l + 1 == r) {
5a4       ++l;
ff5       T ret = ft[r--];
32e       for(l -= lsb(l); l != r; r -= lsb(r)){
78f         ret -= ft[r];
330       }
edf       return ret;
8d7     }
ef7     return query(r) - query(l);
1e9   }
      // Returns largest r such that pred(query(0, r)) == true (or n if none)
851   template <typename Pred>
6ba   int find_right(Pred&& pred) const {
a70     T prefix{};
bec     int pos = 0;
78a     for (int x = __lg(n); x >= 0; --x) {
4a7       int npos = pos + (1 << x);
f5f       if (npos > n) {
5e2         continue;
e2c       }
c48       T nprefix = prefix + ft[npos];
670       if (pred(nprefix)) {
7e5         pos = npos;
e3d         prefix = nprefix;
437       }
f6a     }
d75     return pos;
490   }
3f7   int lower_bound(T value){
ef4     return find_right([value](T x){ return x < value; });
e75   }
260 };
```

5.2 Fenwick Tree 2D

```
67a template<typename T>
2e6 struct FenwickTree2D {
673 public:
2d9   FenwickTree2D(const vector<pair<T, T>>& p) {
790     for(auto [x, _] : p) {
422       ord.push(x);
5dc     }
255     fw.resize(ord.build() + 1);
303     coord.resize((int)fw.size());
5a2     for(auto [x, y] : p) {
213       for(int on = ord[x + 1]; on < (int)fw.size(); on += lsb(on)) {
ee3         coord[on].push(y);
a02       }
646     }
049     for(int i = 0; i < (int)fw.size(); ++i) {
```

```
afd       fw[i].assign(coord[i].build() + 1, T());
3f2     }
468   }
e78   void update(T x, T y, T v) {
a08     for(int xx = ord[x + 1]; xx < (int)fw.size(); xx += lsb(xx)) {
165       for(int yy = coord[xx][y + 1]; yy < (int)fw[xx].size(); yy +=
    lsb(yy)) {
060         fw[xx][yy] += v;
9d1       }
cdf     }
736   }
5d2   T query(T x, T y) {
11f     T ans{};
e3c     for(int xx = ord[x]; xx > 0; xx -= lsb(xx)) {
f00       for(int yy = coord[xx][y]; yy > 0; yy -= lsb(yy)) {
147         ans += fw[xx][yy];
e91       }
335     }
ba7     return ans;
8ed   }
46d   T query(T x1, T y1, T x2, T y2) { // [x1, x2), [y1, y2)
dc1     return query(x2, y2) - query(x2, y1) - query(x1, y2) + query(x1, y1);
c86   }
a86   void update(T x1, T y1, T x2, T y2, T v) {
1fa     update(x1, y1, +v);
2d6     update(x1, y2, -v);
8a7     update(x2, y1, -v);
93d     update(x2, y2, +v);
65b   }
bf2 private:
016   CoordinateCompression<T> ord;
f5a   vector<CoordinateCompression<T>> coord;
7fa   vector<vector<T>> fw;
79d   static int lsb(int b) { return b & -b; }
4fc };
```

5.3 Minimum Cartesian Tree

```
// Arvore de minimos, raiz tem o minimo global, l e r apontam pra posicao dos
    minimos na esq e dir
bf2 struct Node {
4ad   int l, r, p;
a4d   Node(int l = 0, int r = 0, int p = 0) : l(l), r(r), p(p) {}
1d9 };
bae for(int i = 0; i < n; ++i) {
178   tree[i] = Node(-1, -1, i - 1);
201   while(tree[i].p != -1 && h[tree[i].p] > h[i]) {
b18     tree[i].l = tree[i].p;
683     tree[i].p = tree[tree[i].p].p;
15d   }
430   if(tree[i].l != -1) tree[tree[i].l].p = i;
bcf   if(tree[i].p != -1) tree[tree[i].p].r = i;
437 }
```

5.4 Monoid Queue

```
2a1 template<typename T, typename F>
b11 struct MonoidQueue {
8d3   deque<T> q;
bbc   deque<pair<T, int>> m;
5a5   F f;
20f   MonoidQueue(F f = F()) : f(f) {}
3a7   void push(const T& x) {
353     int last_min_dist = m.empty() ? 0 : 1;
310     while(!m.empty() && f(x, m.back().first)) {
ea9       last_min_dist += m.back().second;
4fc       m.pop_back();
4f9     }
cdf     q.emplace_back(x);
08d     m.emplace_back(x, last_min_dist);
771   }
42d   void pop() {
d8a     if(q.front() == m.front().first) {
867       m.pop_front();
e15     }
ced     q.pop_front();
0cd     if(!m.empty()) {
c24       m.front().second -= 1;
b4a     }
213   }
7dd   T front() const { return q.front(); }
      // return min / max value and its position
7c0   pair<T, int> get_extremum() const { return m.front(); }
b55   bool empty() const { return q.empty(); }
d77 };
67a template<typename T>
1c7 using MinQueue = MonoidQueue<T, std::less<T>>;
```

5.5 Segment Tree Lazy

```
baf struct LazyContext {
3ec   int x;
4da   bool is_empty;
e8d   LazyContext() : x(0), is_empty(true) {} // neutral element
17a   LazyContext(int x) : x(x), is_empty(false) {}
835   void compose(const LazyContext& rhs) {
        /* addition to *this */
8f9     is_empty &= rhs.is_empty;
cf2   }
2e5   bool empty() const { return is_empty; }
e64   void reset() { *this = LazyContext(); }
00f };
bf2 struct Node {
3ec   int x;
d63   Node() : x(0) {} // neutral element
4ed   Node(int x) : x(x) {}
```

```
928   Node& operator+=(const Node& rhs) {
        /* addition to *this */
357     return *this;
9d0   }
5fd   friend Node operator+(Node lhs, const Node& rhs) {
705     return lhs += rhs;
950   }
b15   void apply(const LazyContext& lazy) {
        /* update node with lazy */
e02   }
753 };
89d template<typename T, typename L>
2ba struct LazySegmentTree {
673 public:
493   LazySegmentTree(int n = 0) : n(n), st(4 * n, T()), lazy(4 * n, L()) {}
1f7   template<typename Iterator>
bf1   LazySegmentTree(Iterator first, Iterator last) :
    LazySegmentTree(int(last - first)) {
cc9     build(1, 0, n, first);
469   }
ada   void update(int l, int r, const L& val) {
2d5     update(1, 0, n, l, r, val);
340   }
b7a   T query(int l, int r) {
bc4     T cur{};
359     query(1, 0, n, l, r, cur);
75e     return cur;
67b   }
bf2 private:
1a8   int n;
b70   vector<T> st;
672   vector<L> lazy;
309   static int left (int p) { return 2 * p; }
79c   static int right (int p) { return 2 * p + 1; }
1f7   template<typename Iterator>
dbe   void build(int p, int tl, int tr, Iterator first) {
43f     if(tl + 1 == tr) {
6ca       st[p] = first[tl];
2be     } else {
27b       int mid = (tl + tr) / 2;
223       build(left(p), tl, mid, first);
f6a       build(right(p), mid, tr, first);
167       st[p] = st[left(p)] + st[right(p)];
d3f     }
896   }
ad4   void update(int p, int tl, int tr, int l, int r, const L& val) {
dbf     if(tl >= r || tr <= l) {
505       return;
2c6     } else if(tl >= l && tr <= r) {
2e1       st[p].apply(val);
097       lazy[p].compose(val);
869     } else {
b7b       push(p);
27b       int mid = (tl + tr) / 2;
dc8       update(left(p), tl, mid, l, r, val);
3d1       update(right(p), mid, tr, l, r, val);
167       st[p] = st[left(p)] + st[right(p)];
0cd     }
369   }
57c   void query(int p, int tl, int tr, int l, int r, T& cur) {
dbf     if(tl >= r || tr <= l) {
505       return;
2c6     } else if(tl >= l && tr <= r) {
68e       cur += st[p];
1b5     } else {
b7b       push(p);
27b       int mid = (tl + tr) / 2;
b55       query(left(p), tl, mid, l, r, cur);
0f8       query(right(p), mid, tr, l, r, cur);
c3b     }
850   }
3b4   void push(int p) {
81c     if(lazy[p].empty()) {
505       return;
fae     }
2d8     for(int q : {left(p), right(p)}) {
cf4       st[q].apply(lazy[p]);
31c       lazy[q].compose(lazy[p]);
b47     }
f97     lazy[p].reset();
93d   }
410 };
```

5.6 Segment Tree Persistent

```
bf2 struct Node {
e29   int v = 0;
d29   Node *l = this, *r = this;
11d };
c27 const int MS = 1e5;
dc5 Node buffer[20 * MS]; // memory allocation for the nodes;
6c6 Node* root[MS + 1]; // root[i] - pointer to the root of version i
e43 int new_node_cnt = 0;
e49 Node* update(Node* on, int tl, int tr, int x, int val) {
fbb   Node* node = &buffer[new_node_cnt++];
120   *node = *on;
43f   if(tl + 1 == tr) {
089     node->v = val;
192     return node;
35d   } else {
27b     int mid = (tl + tr) / 2;
1dc     if(x < mid) {
bd8       node->l = update(on->l, tl, mid, x, val);
a8b     } else {
fed       node->r = update(on->r, mid, tr, x, val);
a6b     }
4f5     node->v = node->l->v + node->r->v;
192     return node;
```

```
0d3   }
392 }
f21 int query(Node* on,int tl, int tr, int l, int r) {
dbf   if(tl >= r || tr <= l) {
bb3     return 0;
d00   } else if(tl >= l && tr <= r) {
461     return on->v;
e09   } else {
27b     int mid = (tl + tr) / 2;
665     return query(on->l, tl, mid, l, r) + query(on->r, mid, tr, l, r);
6b3   }
115 }
```

5.7 Sparse Table

```
2a1 template<typename T, typename F>
7e9 struct SparseTable {
1a8   int n;
bb5   vector<vector<T>> st;
4a7   vector<int> lg;
5a5   F f;
1f7   template<typename Iterator>
43a   SparseTable(Iterator first, Iterator last, F f = F()) : n(int(last -
    first)), lg(n + 1), f(f) {
8c0     for(int j = 2; j <= n; ++j) {
b83       lg[j] = 1  + lg[j >> 1];
002     }
3fb     st.assign(lg[n] + 1, vector<T>(n));
bf7     copy(first, last, begin(st[0]));
49c     for(int j = 0; j < lg[n]; ++j) {
878       for(int i = 0; i + (1 << (j + 1)) <= n; ++i) {
683         st[j + 1][i] = f(st[j][i], st[j][i + (1 << j)]);
94d       }
38b     }
449   }
6a1   T query(int l, int r) const {
8fd     int j = lg[r - l];
bf1     return f(st[j][l], st[j][r - (1 << j)]);
b50   }
af2 };
67a template <typename T>
49d struct MinFunctor {
6a6   T operator()(const T& x, const T& y) const { return min(x, y); }
c33 };
67a template <typename T>
2b6 using RMQ = SparseTable<T, MinFunctor<T>>;
```

5.8 Treap

```
9c9 const int seed = (int)
    std::chrono::steady_clock::now().time_since_epoch().count();
817 std::mt19937 rng(seed);
350 struct Data {
746   int pref, best, suf;
735   Data(int pref = 0, int best = 0, int suf = 0) : pref(pref), best(best),
    suf(suf) {}
9b8 };
bf2 struct Node {
af7   bool val;
244   int prior, size;
c17   Node *l, *r;
b21   Data data[2];
22a   bool flip_lazy;
2dd   Node() {}
4bf   Node(bool val) : val(val), prior(uniform_int_distribution<>()(rng)),
    size(1), flip_lazy(false) {
a37     data[val] = Data(1, 1, 1);
fb2     data[!val] = Data(0, 0, 0);
222     l = r = nullptr;
07c   }
9b8 };

a3e int size(Node* t) { return t ? t->size : 0; }
144 Data data(Node* t, int e) { return t ? t->data[e] : Data(0, 0, 0); }

c39 void fix(Node* t) {
a26   if(!t) return;
c02   t->size = 1 + size(t->l) + size(t->r);
28d   for(int e = 0; e < 2; ++e) {
116     auto ld = data(t->l, e);
e8c     auto rd = data(t->r, e);
189     t->data[e].pref = ld.pref;
5cf     if(t->data[e].pref == size(t->l)) {
333       t->data[e].pref += t->val == e ? 1 + rd.pref : 0;
887     }
538     t->data[e].suf = rd.suf;
52a     if(t->data[e].suf == size(t->r)) {
55f       t->data[e].suf += t->val == e ? 1 + ld.suf : 0;
811     }
ec2     t->data[e].best = max({ld.best, rd.best, t->data[e].pref,
    t->data[e].suf, t->val == e ? 1 + rd.pref + ld.suf : 0});
3e0   }
223 }
fd8 void apply_flip(Node* t) {
a26   if(!t) return;
0cf   t->val = !t->val;
1e6   swap(t->data[0], t->data[1]);
118 }

998 void push(Node* t) {
a26   if(!t) return;
013   if(!t->flip_lazy) return;
93a   if(t->l) {
7e5     apply_flip(t->l);
```

```
bef     t->l->flip_lazy = !t->l->flip_lazy;
a26   }
d13   if(t->r) {
a05     apply_flip(t->r);
78a     t->r->flip_lazy = !t->r->flip_lazy;
01c   }
1f9   t->flip_lazy = false;
4da }

076 pair<Node*, Node*> split(Node* t, int k) {
987   if(!t) return {};
073   push(t);
c90   if(size(t->l) >= k) {
670     auto [L, R] = split(t->l, k);
307     t->l = R; fix(t); return {L, t};
7eb   } else {
6c5     auto [L, R] = split(t->r, k - size(t->l) - 1);
67b     t->r = L; fix(t); return {t, R};
1a4   }
e23 }

768 Node* merge(Node* l, Node* r) {
df9   if(!l || !r) return l ? l : r;
b8e   push(l); push(r);
da7   if(l->prior > r->prior) {
ed7     l->r = merge(l->r, r);
353     return fix(l), l;
3ac   } else {
654     r->l = merge(l, r->l);
0ac     return fix(r), r;
38d   }
bf3 }

5ef void update(Node* &t, int l, int r) {
e6d   Node *left, *mid, *right;
e11   tie(mid, right) = split(t, r);
648   tie(left, mid) = split(mid, l);
c8b   if(mid) {
f2c     apply_flip(mid);
1ae     mid->flip_lazy = !mid->flip_lazy;
97d   }
151   t = merge(merge(left, mid), right);
d99 }
af0 Node* root = nullptr;
fba for(int i = 0; i < N; ++i) {
7de   root = merge(root, new Node(str[i] == '1'));
9d1 }
// root <- informacao de todo o array
```

6 Math

6.1 Chinese Remainder Theorem

```
67a template <typename T>
788 T extended_gcd(T a, T b, T& x, T& y) {
220   if (a == 0) {
92f     x = 0, y = 1;
73f     return b;
32a   } else {
3a2     T q = b / a, r = b % a;
6b3     T g = extended_gcd(r, a, y, x);
67e     x -= q * y;
96b     return g;
efe   }
708 }
67a template<typename T>
b81 T mul(T a, T b, T m) {
7d4   T q = (long double) a * b / m;
fb4   T r = a * b - q * m;
b8b   return (r + m) % m;
285 }
67a template <typename T>
639 struct CRT {
663   T a, mod;
5c1   CRT() : a(0), mod(1) {}
1db   CRT(T a_, T mod_) : a(a_), mod(mod_) {
67c     a %= mod;
3f8     if (a < 0) a += mod;
872   }
f0d   CRT operator+(CRT rhs) const {
645     T x, y;
ce3     T g = extended_gcd(mod, rhs.mod, x, y);
672     if (a == -1 || rhs.a == -1 || (a - rhs.a) % g) {
6c5       CRT res;
5ed       res.a = -1;
b50       return res;
709     }
f01     T lcm = mod / g * rhs.mod;
4c9     return CRT(a + mul(mul(mod, x, lcm), (rhs.a - a) / g, lcm), lcm);
81e   }
9c6 };
```

6.2 Combinatorics

```
67a template<typename T>
a8f struct Combinatorics {
af1   vector<T> fat, inv, pref, suf;
364   Combinatorics(int n) : fat(n), inv(n), pref(n), suf(n) {
2d5     fat[0] = inv[0] = 1;
6fa     for(int i = 1; i < n; ++i) {
```

```
61d       fat[i] = i * fat[i - 1];
109       inv[i] = 1 / fat[i];
5dd     }
337   }
6a7   T operator()(int n, int k) const {
8b9     return k < 0 || n < k ? 0 : fat[n] * inv[k] * inv[n - k];
da9   }
      // interpolate points (i, y[i]) and evaluate at x
7f8   T interpolate(const vector<T>& y, T x) {
3ac     int n = (int)y.size();
1b8     pref[0] = suf[n - 1] = 1;
7a4     for(int i = 0; i + 1 < n; ++i) {
c44       pref[i + 1] = pref[i] * (x - i);
82d     }
ac5     for(int i = n - 1; i > 0; --i) {
ceb       suf[i - 1] = suf[i] * (x - i);
d82     }
966     T ans = 0;
        // beware negative sgn
dfe     for(int i = 0, sgn = (n % 2 ? +1 : -1); i < n; ++i, sgn *= -1) {
46d       ans += sgn * y[i] * pref[i] * suf[i] * inv[i] * inv[n - 1 - i];
3c4     }
ba7     return ans;
ec9   }
dc7 };
```

6.3 Convolution - FFT

```
3af using Complex = complex<double>;
722 void fft(vector<Complex>& a) {
f82   static vector<complex<long double>> R(2, 1);
fac   static vector<Complex> rt(2, 1); // (^ 10% faster if double)
1b8   int n = (int)a.size(), L = 31 - __builtin_clz(n);
ad8   for(static int k = 2; k < n; k *= 2) {
9d9     R.resize(n);
335     rt.resize(n);
411     auto x = polar(1.0L, acos(-1.0L) / k);
bc3     for (int i = k; i < 2 * k; ++i) {
cd4       rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
bc8     }
b91   }
808   vector<int> rev(n);
bae   for(int i = 0; i < n; ++i) {
fd9     rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
a75     if (i < rev[i]) swap(a[i], a[rev[i]]);
3ae   }
657   for (int k = 1; k < n; k *= 2)
1e5     for (int i = 0; i < n; i += 2 * k)
3c5       for (int j = 0; j < k; ++j) {
            // Complex z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)
                /// include-line
830           auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k];
    /// exclude-line
2ec           Complex z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
    /// exclude-line
20a           a[i + j + k] = a[i + j] - z;
1b0           a[i + j] += z;
cef         }
212 }

8c8 template<typename T = long long>
510 vector<T> convolution(const vector<T>& a, const vector<T>& b) {
f88   if (a.empty() || b.empty()) return {};
179   vector<T> res((int)a.size() + (int)b.size() - 1);
a2f   int L = 32 - __builtin_clz((int)res.size()), n = 1 << L;
487   vector<Complex> in(n), out(n);
3c3   copy(a.begin(), a.end(), in.begin());
e6e   for (int i = 0; i < (int)b.size(); ++i) in[i].imag(b[i]);
21a   fft(in);
427   for (Complex& x: in)  x *= x;
efe   for (int i = 0; i < n; ++i) out[i] = in[-i & (n - 1)] - conj(in[i]);
3d7   fft(out);
49c   for (int i = 0; i < (int)res.size(); ++i) {
e70     res[i] = (T)llround(imag(out[i]) / (4 * n)); // remove rounding if the
    output is double
283   }
b50   return res;
442 }
```

6.4 Convolution - FFT MOD

```
b66 template<const int MOD, typename T = long long>
225 vector<T> convolution_mod(const vector<T>& a, const vector<T>& b) {
f88   if (a.empty() || b.empty()) return {};
179   vector<T> res((int)a.size() + (int)b.size() - 1);
dcc   int B = 32 - __builtin_clz((int)res.size()), n = 1 << B, cut =
    int(sqrt(MOD));
775   vector<Complex> L(n), R(n), outs(n), outl(n);
828   for(int i = 0; i < (int)a.size(); ++i) L[i] = Complex((int)a[i] / cut,
    (int)a[i] % cut);
d80   for(int i = 0; i < (int)b.size(); ++i) R[i] = Complex((int)b[i] / cut,
    (int)b[i] % cut);
5d5   fft(L), fft(R);
bae   for(int i = 0; i < n; ++i) {
39d     int j = -i & (n - 1);
65e     outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
91a     outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
e30   }
d08   fft(outl), fft(outs);
49c   for(int i = 0; i < (int)res.size(); ++i) {
52d     T av = (T)llround(real(outl[i])), cv = (T)llround(imag(outs[i]));
0d1     T bv = (T)llround(imag(outl[i])) + (T)llround(real(outs[i]));
8d0     res[i] = ((av % MOD * cut + bv) % MOD * cut + cv) % MOD;
a03   }
b50   return res;
698 }
```

6.5 Convolution - NTT

```
e31 const uint32_t MOD = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
29b template<typename Mint>
9a1 void ntt(vector<Mint> &a) {
1b8   int n = (int)a.size(), L = 31 - __builtin_clz(n);
09d   static vector<Mint> rt(2, 1);
8ee   for (static int k = 2, s = 2; k < n; k *= 2, s++) {
335     rt.resize(n);
514     Mint z[] = {1, bin_exp(Mint(root), MOD >> s)};
d34     for(int i = k; i < 2 * k; ++i) rt[i] = rt[i / 2] * z[i & 1];
72f   }
808   vector<int> rev(n);
bae   for(int i = 0; i < n; ++i) {
fd9     rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
a75     if(i < rev[i]) swap(a[i], a[rev[i]]);
3ae   }
657   for (int k = 1; k < n; k *= 2)
1e5     for (int i = 0; i < n; i += 2 * k)
3c5       for(int j = 0; j < k; ++j) {
a4f         Mint z = rt[j + k] * a[i + j + k], &ai = a[i + j];
73d         a[i + j + k] = ai - z;
584         ai += z;
76c       }
c63 }
29b template<typename Mint>
102 vector<Mint> convolution(const vector<Mint> &a, const vector<Mint> &b) {
f88   if (a.empty() || b.empty()) return {};
f65   int s = (int)a.size() + (int)b.size() - 1, B = 32 - __builtin_clz(s), n
    = 1 << B;
f63   Mint inv = 1 / Mint(n);
ac5   vector<Mint> L(a), R(b), out(n);
6b4   L.resize(n), R.resize(n);
d9e   ntt(L), ntt(R);
c14   for(int i = 0; i < n; ++i) out[-i & (n - 1)] = L[i] * R[i] * inv;
ec9   ntt(out);
308   out.resize(s);
fe8   return out;
90c }
```

6.6 Euler Totient

```
c75 int phi[LIM];
8e0 void sieve() {
9a6   iota(phi, phi + LIM, 0);
48b   for(int i = 2; i < LIM; ++i) {
e32     if(phi[i] == i) {
ebc       for(int j = i; j < LIM; j += i) {
a9b         phi[j] -= phi[j] / i;
4bc       }
d20     }
bcb   }
bcd }
67a template<typename T>
e6f T phi(T n) {
fc4   T ans = n;
3f0   for(T p = 2; p * p <= n; ++p) {
80a     if(n % p == 0) {
b7f       ans -= ans / p;
03e       while(n % p == 0) {
f4a         n /= p;
91f       }
d76     }
8fb   }
b26   if(n > 1) {
675     ans -= ans / n;
c1b   }
ba7   return ans;
9bd }
```

6.7 Gaussian Elimination

```
67a template<typename T>
02a struct GaussianElimination {
      // may change if using doubles
e1c   static bool cmp(const T& a, const T& b) { return a == b; }
741   vector<vector<T>> a, inv;
d5f   vector<int> pivot;
532   GaussianElimination(const vector<vector<T>> a = {}) : a(a) {}
9bc   void add_equation(const vector<T>& equation) {
769     a.emplace_back(equation);
a7a   }
      /*
41f     pair(0, ans) impossible
9be     pair(1, ans) one solution
d3e     pair(2, ans) infinite solutions
c4c   */
93a   pair<int, vector<T>> solve_system(bool findInverse = false) {
8ec     int n = (int)a.size();
ae2     int m = (int)a[0].size() - 1;
5b0     pivot.assign(m, -1);
d65     if(findInverse) {
bc5       inv.assign(n, vector<T>(n));
aea       for(int i = 0; i < n; ++i) inv[i][i] = T(1);
d8e     }
61e     for(int col = 0, row = 0; col < m && row < n; ++col) {
0a8       int sel = -1;
0c0       for(int i = row; i < n; ++i) {
e10         if(!cmp(a[i][col], 0)) {
403           sel = i;
c2b           break;
f7b         }
3cd       }
```

```
c79       if(sel == -1) continue;
563       for(int j = col; j <= m; ++j) {
1ec         swap(a[row][j], a[sel][j]);
82c       }
f84       if(findInverse) swap(inv[row], inv[sel]);
bae       for(int i = 0; i < n; ++i) {
d97         if(i == row) continue;
96c         T c = a[i][col] / a[row][col];
563         for(int j = col; j <= m; ++j) {
d34           a[i][j] -= c * a[row][j];
790         }
925         if(!findInverse) continue;
859         for(int j = 0; j < n; ++j) {
d90           inv[i][j] -= c * inv[row][j];
84a         }
084       }
5cc       pivot[col] = row++;
edd     }
9e8     vector<T> ans(m);
544     for(int j = 0; j < m; ++j) {
a34       if(pivot[j] == -1) continue;
          // normalize pivots
851       int i = pivot[j];
296       for(int k = j + 1; k <= m; ++k) {
fc7         a[i][k] /= a[i][j];
c91       }
d65       if(findInverse) {
5e4         for(int k = 0; k < n; ++k) {
912           inv[i][k] /= a[i][j];
968         }
062       }
7fd       a[i][j] = T(1);
697       ans[j] = a[i][m];
ef4     }
bae     for(int i = 0; i < n; ++i) {
b44         T value(0);
544         for(int j = 0; j < m; ++j) {
460           value += ans[j] * a[i][j];
f5c         }
d66         if(!cmp(value, a[i][m])) return make_pair(0, ans);
834     }
544     for(int j = 0; j < m; ++j) {
5ec       if(pivot[j] == -1) return make_pair(2, ans);
869     }
e34     return make_pair(1, ans);
fae   }
9b2 };
```

6.8 Lowest Prime

```
189 int lp[ms]; // lp[i]: lowest prime of i
8c3 vector<int> pr;
8e0 void sieve() {
746   for(int i = 2; i < ms; ++i) {
719     if(lp[i] == 0) {
116       lp[i] = i;
bc0       pr.push_back(i);
d4f     }
38e     for(int p : pr) {
0d5       if(p > lp[i] || i * p >= ms)break;
775       lp[i * p] = p;
f51     }
8fe   }
758 }
67a template<typename T>
419 void get_primes(int x, T&& get) {
061   while(x != 1) {
fe7     int p = lp[x];
4c0     int e = 0;
fa7     while(x%p==0) {
43f       x /= p;
95c       ++e;
b54     }
a47     get(p, e);
8ed   }
cc3 }
13b vector<int> get_divisors(int x) {
6e7   vector<int> divisors({1});
061   while(x != 1) {
3a3     get_primes(x, [&x, &divisors](int p, int e) {
c72       int n = (int)divisors.size();
bae       for(int i = 0; i < n; ++i) {
fa3         int u = divisors[i];
256         for(int j = 0, v = p; j < e; ++j, v *= p) {
fe8           divisors.emplace_back(u * v);
db6         }
54f       }
65a       for(int j = 0; j < e; ++j) x /= p;
504     });
3cb   }
e83   return divisors;
d94 }
667 vector<int> get_masks(int x) {
fd7   vector<int> masks({1});
061   while(x != 1) {
13f     get_primes(x, [&x, &masks](int p, int e) {
467       int n = (int)masks.size();
bae       for(int i = 0; i < n; ++i) {
21d         int u = masks[i];
525         masks.emplace_back(u * p);
e3a       }
65a       for(int j = 0; j < e; ++j) x /= p;
f87     });
c90   }
3c8   return masks;
a80 }
```

6.9 Miller Rabin

```
f85 ull modmul(ull a, ull b, ull M) {
2dd   ll ret = a * b - M * ull(1.L / M * a * b);
964   return ret + M * (ret < 0) - M * (ret >= (ll)M);
e93 }
4f6 ull modpow(ull b, ull e, ull mod) {
c1a   ull ans = 1;
b59   for (; e; b = modmul(b, b, mod), e /= 2) {
eee     if (e & 1) {
bc6       ans = modmul(ans, b, mod);
ea3     }
ab2   }
ba7   return ans;
236 }
87c bool is_prime(ull n) {
84d   if (n < 2 || n % 6 % 4 != 1) {
f64     return (n | 1) == 3;
6b1   }
062   ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
ae0   ull s = __builtin_ctzll(n - 1), d = n >> s;
e80   for (ull a : A) {   // ^ count trailing zeroes
6b4     ull p = modpow(a % n, d, n), i = s;
6d0     while (p != 1 && p != n - 1 && a % n && i--) {
c77       p = modmul(p, p, n);
cfa     }
f85     if (p != n-1 && i != s) {
bb3       return 0;
25c     }
cd9   }
6a5   return 1;
23d }

7eb ull pollard(ull n) {
c3c   auto f = [n](ull x) { return modmul(x, x, n) + 1; };
222   ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
f51   while (t++ % 40 || gcd(prd, n) == 1) {
663     if (x == y) {
2f2       x = ++i, y = f(x);
1e8     }
e11     if ((q = modmul(prd, max(x,y) - min(x,y), n))) {
629       prd = q;
0f8     }
b78     x = f(x), y = f(f(y));
f8f   }
002   return gcd(prd, n);
dc7 }
591 vector<ull> factor(ull n) {
e3b   if (n == 1) {
21d     return {};
a99   }
a7a   if (is_prime(n)) {
48e     return {n};
4c9   }
bc6   ull x = pollard(n);
52a   auto l = factor(x), r = factor(n / x);
98a   l.insert(end(l), begin(r), end(r));
792   return l;
395 }
```

6.10 Mobius Function

```
6f5 int mu[LIM];
cae bool pr[LIM];
53a vector<int> mask[LIM];
1db void build_mobius() {
bac   mu[1] = 1;
3ef   for(int i = 1; i < LIM; ++i) {
cc2     for(int j = 2 * i; j < LIM; j += i) {
6b8       mu[j] -= mu[i];
45a     }
07d   }
06d }
8e0 void sieve() {
3b2   fill(mu, mu + LIM, 1);
e78   memset(pr, 1, sizeof(pr));
48b   for(int i = 2; i < LIM; ++i) {
017     if(pr[i]) {
31d       mu[i] = -1;
bcf       for(int j = i + i; j < LIM; j += i) {
194         mu[j] *= -1;
d1b         pr[j] = false;
a41       }
0f6       if(LIM / i / i == 0) {
5e2         continue;
995       }
4b4       for(int sq = i * i, j = sq; j < LIM; j += sq) {
012         mu[j] = 0;
ba7       }
c91     }
273   }
e93 }
559 void build_mask() {
48b   for(int i = 2; i < LIM; ++i) {
c48     if(mu[i] != 0) {
ebc       for(int j = i; j < LIM; j += i) {
51e         mask[j].emplace_back(i);
3cc       }
0ee     }
63f   }
4f8 }
```

6.11 Modular Arithmetic

```
67a template<typename T>
```

```
56c T bin_exp(T a, long long e) {
dac   T r(1);
d0e   for(; e > 0; e >>= 1) {
eee     if(e & 1) {
1c8       r *= a;
d4b     }
70c     a *= a;
ef5   }
4c1   return r;
d51 }
016 template<const uint32_t MOD>
bb6 struct Mod {
622   uint32_t x;
77d   Mod() : x(0) {};
67a   template<typename T>
ea0   Mod(T x) : x(uint32_t(((int64_t(x) % MOD) + MOD) % MOD)) {}
ecc   Mod& operator+=(Mod rhs) {
393     x += rhs.x;
290     if(x >= MOD) x -= MOD;
357     return *this;
7f3   }
1bd   Mod& operator-=(Mod rhs) {
c2b     x += MOD - rhs.x;
290     if(x >= MOD) x -= MOD;
357     return *this;
51d   }
ead   Mod& operator*=(Mod rhs) {
4e6     auto y = 1ull * x * rhs.x;
2aa     if(y >= MOD) y %= MOD;
a6e     x = uint32_t(y);
357     return *this;
89a   }
4b8   Mod& operator/=(Mod rhs) { return *this *= bin_exp(rhs, MOD - 2); }
ce9   friend Mod operator+(Mod lhs, Mod rhs) { return lhs += rhs; }
16b   friend Mod operator-(Mod lhs, Mod rhs) { return lhs -= rhs; }
d5c   friend Mod operator*(Mod lhs, Mod rhs) { return lhs *= rhs; }
5b7   friend Mod operator/(Mod lhs, Mod rhs) { return lhs /= rhs; }
2b2   bool operator==(Mod rhs) const { return x == rhs.x; }
17e   friend ostream& operator<<(ostream& os, const Mod& o) { return os <<
    o.x; }
52f   friend istream& operator>>(istream& is, Mod& o) {
c23     int64_t x;
af7     is >> x;
84c     o = Mod(x);
fed     return is;
f1b   }
5a9 };
```

6.12 Multiplicative Function

```
67a template<typename T>
7a5 struct MultiplicativeFunction{
2da   vector<T> ans;
a14   vector<bool> pr;
      //Dirichlet == true: unit function (ans[1] = 1, ans[i] = 0)
      //Dirichlet == false: constant function (ans[i] = 1)
c3a   MultiplicativeFunction(int n, bool Dirichlet = true) : ans(n) {
a81     if(Dirichlet) ans[1] = 1;
278     else fill(begin(ans), end(ans), 1);
450   }
      //f: evaluates the multiplicative function at a prime power
398   template<typename F>
f73   MultiplicativeFunction(int n, F&& f):ans(n, 1), pr(n, 1){
9d3     pr[1] = false;
490     for(int i = 2; i < n; ++i){
494       if(!pr[i]) continue;
549       ans[i] = f(i, 1);
3e0       for(int u = i, e = 2; u < n / i; u *= i, ++e) {
180         ans[u * i] = f(i, e);
2df       }
827       for(int j = i + i; j < n; j += i) {
5da         int x = j;
a69         while(x % i == 0) x /= i;
d04         ans[j] = ans[x] * ans[j / x]; // multiplicative property: (x, j /
    x) = 1
d1b         pr[j] = false;
952       }
2da     }
98f   }
771   using MF = MultiplicativeFunction<T>;
      // Dirichlet convolution
      // f * g [n] = sum of f[d] * g[n/d]
9e8   MF& operator*=(const MF& rhs) {
d29     int n = (int)ans.size();
8ec     vector<T> r(n);
8e7     for(int i = 1; i < n; ++i)
1ec       for(int j = i; j < n; j += i)
7ae         r[j] += ans[i] * rhs[j / i];
3be     ans.swap(r);
357     return *this;
17c   }
e71   friend MF operator*(MF lhs, const MF& rhs) { return lhs *= rhs; }
73b   const T& operator[](int i) const { return ans[i]; }
e7d };
67a template<typename T>
771 using MF = MultiplicativeFunction<T>;
67a template<typename T>
803 MF<T> bin_exp(MF<T> a, long long e) {
eaa   int n = (int)a.ans.size();
23a   MF<T> r(n, true);
d0e   for(; e > 0; e >>= 1) {
442     if(e & 1) r *= a;
70c     a *= a;
216   }
4c1   return r;
afb }
// Mobius function
67a template<typename T>
```

```
389 struct Mobius : MF<T> {
898   using MF<T>::MF;
957   using MF<T>::operator=;
a2b   Mobius(int n) : MF<T>(n, [](int, int e) {
a67     return e > 1 ? 0 : -1;
758   }) {};
502 };
// Euler's totient
67a template<typename T>
46a struct PHI : MF<T> {
898   using MF<T>::MF;
957   using MF<T>::operator=;
5e0   PHI(int n) : MF<T>(n, [](int p, int e){
55e     T pw = 1;
70f     for(int j = 0; j < e - 1; ++j) pw *= p;
da7     return pw * (p - 1);
758   }) {};
553 };
// Number of divisors
67a template<typename T>
d9a struct NUMDIV : MF<T> {
898   using MF<T>::MF;
957   using MF<T>::operator=;
54a   NUMDIV(int n) : MF<T>(n, [](int, int e){
350     return e + 1;
758   }) {};
e95 };
// Sum of divisors
67a template<typename T>
16e struct SUMDIV : MF<T> {
898   using MF<T>::MF;
957   using MF<T>::operator=;
7ef   SUMDIV(int n):MF<T>(n, [](int p, int e){
55e     T pw = 1;
6f1     for(int j = 0; j < e + 1; ++j) pw *= p;
b5d     return (pw - 1) / (p - 1);
758   }) {};
55b };
```

6.13 Number of Divisors

```
3fe int num_div[LIM]; //num_div[i] = number of divisors of i
8e0 void sieve() {
3ef   for(int i = 1; i < LIM; ++i) {
ebc     for(int j = i; j < LIM; j += i) {
74c       num_div[j]++;
e51     }
a4c   }
f83 }
```

6.14 Primitive Root

```
b11 const uint32_t MOD = (119 << 23) + 1;
45c using Mint = Mod<MOD>;
// r is a primitive root mod M iff r^k == a mod M for every a gcd(M, a) = 1
    there exists k
fd4 bool primitive_root(int r) {
5cc   int m = MOD - 1;
bc8   for(int i = 2; i * i <= m; ++i) {
75a     if(m % i == 0) {
b3a       if(bin_exp(Mint(r), i) == 1) return false;
c9b       if(bin_exp(Mint(r), m / i) == 1) return false;
322     }
a10   }
8a6   return true;
8a9 }
```

6.15 Static Matrix

```
eb9 const int N = 2;
67a template<typename T>
bf3 struct Matrix {
fe2   T a[N][N];
20c   Matrix(bool identity = false) {
fba     for(int i = 0; i < N; ++i) {
34f       for(int j = 0; j < N; ++j) {
bc5         a[i][j] = T(0);
d63       }
b91       a[i][i] = T(identity);
a71     }
74c   }
98d   Matrix operator *(const Matrix& b) {
0fa     Matrix p;
282     for(int i = 0; i < N; ++i)
75a       for(int j = 0; j < N; ++j)
1e5         for(int k = 0; k < N; ++k)
37e           p.a[i][k] += a[i][j] * b.a[j][k];
74e     return p;
9a4   }
6fc };
```

6.16 Sum of Divisors

```
415 long long sum_div[LIM]; //sum_div[i] =sum of divisors of i
8e0 void sieve() {
3ef   for(int i = 1; i < LIM; ++i) {
cc2     for(int j = 2 * i ; j < LIM; j +=i) {
ed5       sum_div[j] += i;
d7a     }
797   }
```

```
9ea }
d41
```

6.17 Xor Gauss

```
a74 template<const int N, class T = unsigned int>
b94 struct XorGauss {
b71   T basis[N]{};
1fc   int sz = 0;
3d0   T reduce(T x) const {
fcc     for(int i = N - 1; i >= 0; --i) {
9d4       x = std::min(x, x ^ basis[i]);
5d2     }
ea5     return x;
cb1   }
947   T augment(T x) const {
a6f     return ~reduce(~x);
02b   }
4c9   bool add(T x) {
fcc     for(int i = N - 1; i >= 0; --i) {
393       if(((x >> i) & 1) == 0) {
5e2         continue;
b6d       }
122       if(basis[i]) {
6b0         x ^= basis[i];
953       } else {
c6c         basis[i] = x;
6f4         sz += 1;
8a6         return true;
7c4       }
a84     }
d1f     return false;
f8c   }
fc7 };
```

7 Dynamic Programming

7.1 Divide and Conquer DP

```
de5 ll cost(int l, int r) { /* transition cost for l...r (inclusive)*/ }
// dp[i][j] = min_i{0 <= k <= j}(dp[i - 1][k - 1] + cost(k, j))
// condition for it: opt(i, j) <= opt(i, j + 1)
// special case to check: cost(a, c) + cost(b, d) <= cost(a, d) + cost(b, c) a
    <= b <= c <= d
// computes dp[i][l]...dp[i][r] (inclusive)
c03 void divide_and_conquer(int e, int l, int r, int opt_l, int opt_r) {
de6   if(l > r) return;
ae0   int mid = (l + r) / 2;
c58   pair<ll, int> best = {INF, -1};
933   for(int k = opt_l; k <= min(opt_r, mid); ++k) {
59a     ll cur = (k > 0 ? dp[k - 1][e^1] : 0) + cost(k, mid);
8f3     if(cur < best.first) {
5fc       best = {cur, k};
3c4     }
40e   }
64a   dp[mid][e] = best.first;
24a   divide_and_conquer(e, l, mid - 1, opt_l, best.second);
ed7   divide_and_conquer(e, mid + 1, r, best.second, opt_r);
fb1 }
0f4 for(int i = 0; i < n; ++i) dp[i][0] = cost(0, i); // initial cost
01e for(int i = 1; i < k; ++i) divide_and_conquer(i % 2, 0, n - 1, 0, n - 1);

// alternativaly, maintain cost function for [opt_l, r] during recursion
8a0 ll add(ll cost, ll x) { /* update cost adding element x */ }
be4 ll remove(ll cost, ll x) { /* remove cost removing element x */ }

// maintain cost [opt_l, r]
88f void divide_and_conquer(int e, int l, int r, int opt_l, int opt_r, ll
    cost) {
de6   if(l > r) return;
ae0   int mid = (l + r) / 2;
277   pair<ll, int> best = {-INF, 0};
6ab   for(int k = r; k > mid; --k) {
e78     cost = remove(cost, a[k]);
8a5   } // cost [k, mid]
933   for(int k = opt_l; k <= min(opt_r, mid); ++k) {
f95     int cur = (k > 0 ? dp[e ^ 1][k - 1] : 0) + cost;
e78     cost = remove(cost, a[k]);
152     if(cur > best.first) {
5fc       best = {cur, k};
dc7     }
f11   } // cost [min(opt_r, mid) + 1, mid]
35f   dp[e][mid] = best.first;
d76   for(int k = min(opt_r, mid); k >= opt_l; --k) {
9a3     cost = add(cost, a[k]);
44b   }
274   cost = remove(cost, a[mid]); // cost [opt_l, mid - 1]
669   divide_and_conquer(e, l, mid - 1, opt_l, best.second, cost);
49e   for(int k = mid; k <= r; ++k) {
9a3     cost = add(cost, a[k]);
542   }
260   for(int k = opt_l; k < best.second; ++k) {
e78     cost = remove(cost, a[k]);
9d5   }
      // cost [best.second, r]
3e6   divide_and_conquer(e, mid + 1, r, best.second, opt_r, cost);
ef8   for(int k = best.second - 1; k >= opt_l; --k) {
9a3     cost = add(cost, a[k]);
53a   }  // restore cost to [opt_l, r]
e40 }
```

7.2 Line Container

```
// lower hull, max query
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};
struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use INF = 1/.0, div(a,b) = a/b)
  static const ll INF = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) {
      x->p = INF;
      return false;
    }
    if (x->k == y->k) {
      x->p = x->m > y->m ? INF : -INF;
    } else {
      x->p = div(y->m - x->m, x->k - y->k);
    }
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) {
      z = erase(z);
    }
    if (x != begin() && isect(--x, y)) {
      isect(x, y = erase(y));
    }
    while ((y = x) != begin() && (--x)->p >= y->p) {
      isect(x, erase(y));
    }
  }
  ll query(ll x) const {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

7.3 Sack - path to root

```
void usedp(int u, LineContainer& dp) {
  ans[u] = min(ans[u], S[u] + dist[u] * V[u] - dp.query(V[u]));
}
void filllight(int u, int p, LineContainer& dp) {
  usedp(u, dp);
  for(auto [v, _] : adj[u]) {
    if(v == p) continue;
    filllight(v, u, dp);
  }
}
void dfs(int u, int p, LineContainer& dp) {
  dp.add(dist[u], -ans[u]);
  int big = -1;
  for(auto [v, w] : adj[u]) {
    if(v == p) continue;
    if(big == -1 || sz[v] > sz[big]) {
      big = v;
    }
  }
  for(auto [v, w] : adj[u]) {
    if(v == p || v == big) continue;
    filllight(v, u, dp);
    LineContainer nxtdp;
    dfs(v, u, nxtdp);
  }

  if(big != -1) {
    usedp(big, dp);
    dfs(big, u, dp);
  }
}
```

7.4 Sack - subtree

```
void dfs(int u, int p = -1, bool keep = 0) {
  int big = -1;
  for (int v : adj[u]) {
    if (v == p) continue;
    if (big == -1 || sz[v] > sz[big]) {
      big = v;
    }
  }
  for (int v : adj[u]) {
    if (v == p || v == big) continue;
    dfs(v, u, 0);
  }
  if (big != -1) {
    dfs(big, u, 1);
  }
  for (int v : adj[u]) {
    if (v == p || v == big) continue;
    put(v, u);
  }
  if (!keep) {

  }
}
```

8 Extra