



Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1	Graphs	2			
1.1	Lca	2	2.3	Inside Polygon	20
1.2	Hld Vertex	2	2.4	Shoelace Boundary	21
1.3	Hld Edge	4	3	Misc	21
1.4	Tarjan Bridge	5	3.1	Int128	21
1.5	2sat	5	3.2	Split	21
1.6	Dijkstra	7	4	Data Structures	21
1.7	Ford Fulkerson Edmonds Karp	7	4.1	Psum2d	21
1.8	Bipartite	8	4.2	Range Query Point Update	22
1.9	Floyd Warshall	8	4.3	Persistent	23
1.10	Hungarian	8	4.4	Minimum And Amount	23
1.11	Centroid Decomposition	9	4.5	Lazy Assignment To Segment	24
1.12	Tree Diameter	9	4.6	Segtree2d	25
1.13	Kuhn	9	4.7	Dynamic Implicit Sparse	26
1.14	Negative Cycle	10	4.8	Segment With Maximum Sum	27
1.15	Eulerian Undirected	10	4.9	Lazy Addition To Segment	28
1.16	Bellman Ford	11	4.10	Lazy Dynamic Implicit Sparse	29
1.17	Blossom	12	4.11	Sparse Table	30
1.18	Kruskall	12	4.12	Sparse Table2d	31
1.19	Small To Large	13	4.13	Ordered Set	31
1.20	Prim	13	4.14	Priority Queue	32
1.21	Cycle Path Recovery	14	4.15	Two Sets	32
1.22	Min Cost Max Flow	14	4.16	Dsu	32
1.23	Eulerian Directed	15	4.17	Mergesort Tree Ordered Set	33
1.24	Find Cycle	15	4.18	Mergesort Tree Vector	34
1.25	Dinic	16	5	Math	35
1.26	Centroid Find	17	5.1	Crt	35
2	Geometry	18	5.2	Function Root	35
2.1	Closest Pair Points	18	5.3	Prime Factors	35
2.2	2d	18	5.4	Subsets	35
			5.5	To Decimal	35

5.6	Multiplicative Inverse	35
5.7	Set Operations	36
5.8	Representation Arbitrary Base	36
5.9	Matrix Exponentiation	36
5.10	Fast Exponentiation	37
5.11	Phi	37
5.12	Binary To Decimal	37
5.13	Ceil	37
5.14	Horner Algorithm	37
5.15	Mobius	37
5.16	Sieve Of Eratosthenes	38
5.17	Divisors	38
5.18	Linear Diophantine Equation	38
5.19	Check If Bit Is On	39
6	Template	39
6.1	Template	39
6.2	Template Clean	39
7	Algorithms	39
7.1	Delta-encoding	39
7.2	Subsets	39
7.3	Ternary Search	40
7.4	Biggest K	40
7.5	Binary Search First True	40
7.6	Binary Search Last True	40
7.7	Lis	40
8	Strings	41
8.1	Generate All Sequences Length K	41
8.2	Lcs	41
8.3	Hash	41
8.4	Trie	41
8.5	Generate All Permutations	42
8.6	Kmp	42
8.7	Hash2	42
8.8	Suffix Array	42
8.9	Z-function	44
9	DP	44
9.1	Kadane	44
9.2	Edit Distance	44
9.3	Coins	45
9.4	Minimum Coin Change	45
9.5	Substr Palindrome	45
9.6	Digits	45
9.7	Knapsack With Index	45
9.8	Knapsack	46

1 Graphs

1.1 Lca

```
1 // Description:
2 // Find the lowest common ancestor between two nodes
  in a tree
3
4 // Problem:
5 // https://cses.fi/problemset/task/1135
6
7 // Complexity:
8 // O(log n)
9
10 // How to use:
11 // preprocess();
12 // lca(a, b);
13
14 // Notes
15 // To calculate the distance between two nodes use
  the following formula
16 // level_peso[a] + level_peso[b] - 2*level_peso[lca(a
  , b)]
17
18 const int MAX = 2e5+10;
19 const int BITS = 30;
20
21 vector<pii> adj[MAX];
22 vector<bool> visited(MAX);
23
24 int up[MAX][BITS + 1];
25 int level[MAX];
26 int level_peso[MAX];
27
28 void find_level() {
29     queue<pii> q;
30
31     q.push(mp(1, 0));
32     visited[1] = true;
33
34     while (!q.empty()) {
35         auto [v, depth] = q.front();
36         q.pop();
37         level[v] = depth;
38
39         for (auto [u,d] : adj[v]) {
40             if (!visited[u]) {
41                 visited[u] = true;
42                 up[u][0] = v;
43                 q.push(mp(u, depth + 1));
44             }
45         }
46     }
47 }
48
49 void find_level_peso() {
50     queue<pii> q;
51
52     q.push(mp(1, 0));
53     visited[1] = true;
54
55     while (!q.empty()) {
56         auto [v, depth] = q.front();
57         q.pop();
58         level_peso[v] = depth;
59
60         for (auto [u,d] : adj[v]) {
61             if (!visited[u]) {
62                 visited[u] = true;
63                 up[u][0] = v;
64                 q.push(mp(u, depth + d));
65             }
66         }
67     }
68 }
```

```
66     }
67 }
68 }
69
70 int lca(int a, int b) {
71     // get the nodes to the same level
72     int mn = min(level[a], level[b]);
73
74     for (int j = 0; j <= BITS; j++) {
75         if (a != -1 && ((level[a] - mn) & (1 << j))) a
76         = up[a][j];
77         if (b != -1 && ((level[b] - mn) & (1 << j))) b
78         = up[b][j];
79     }
80
81     // special case
82     if (a == b) return a;
83
84     // binary search
85     for (int j = BITS; j >= 0; j--) {
86         if (up[a][j] != up[b][j]) {
87             a = up[a][j];
88             b = up[b][j];
89         }
90     }
91     return up[a][0];
92 }
93
94 void preprocess() {
95     visited = vector<bool>(MAX, false);
96     find_level();
97     visited = vector<bool>(MAX, false);
98     find_level_peso();
99
100     for (int j = 1; j <= BITS; j++) {
101         for (int i = 1; i <= n; i++) {
102             if (up[i][j - 1] != -1) up[i][j] = up[up[i][j - 1]][j - 1];
103         }
104     }
105 }
```

1.2 Hld Vertex

```
1 // Description:
2 // Make queries and updates between two vertexes on a
  tree
3 // Query path - query path (a, b) inclusive
4 // Update path - update path (a, b) inclusive
5 // Query subtree - query subtree of a
6 // Update subtree - update subtree of a
7 // Update - update vertex or edge
8 // Lca - get lowest common ancestor of a and b
9 // Search - perform a binary search to find the last
  node with a certain property
10 // on the path from a to the root
11
12 // Problem:
13 // https://codeforces.com/gym/101908/problem/L
14
15 // Complexity:
16 // O(log ^2 n) for both query and update
17
18 // How to use:
19 // HLD hld = HLD(n + 1, adj)
20
21 // Notes
22 // Change the root of the tree on the constructor if
  it's different from 1
23 // Use together with Segtree
24
25 typedef long long ftype;
26
```

```

27 struct HLD {
28     vector<int> parent;
29     vector<int> pos;
30     vector<int> head;
31     vector<int> subtree_size;
32     vector<int> level;
33     vector<int> heavy_child;
34     vector<ftype> subtree_weight;
35     vector<ftype> path_weight;
36     vector<vector<int>> adj;
37     vector<int> at;
38     Segtree seg = Segtree(0);
39     int cpos;
40     int n;
41     int root;
42     vector<vector<int>> up;
43
44     HLD() {}
45
46     HLD(int n, vector<vector<int>>& adj, int root = 1)
47     : adj(adj), n(n), root(root) {
48         seg = Segtree(n);
49         cpos = 0;
50         at.resize(n);
51         parent.resize(n);
52         pos.resize(n);
53         head.resize(n);
54         subtree_size.assign(n, 1);
55         level.assign(n, 0);
56         heavy_child.assign(n, -1);
57         parent[root] = -1;
58         dfs(root, -1);
59         decompose(root, -1);
60
61     void dfs(int v, int p) {
62         parent[v] = p;
63         if (p != -1) level[v] = level[p] + 1;
64         for (auto u : adj[v]) {
65             if (u != p) {
66                 dfs(u, v);
67                 subtree_size[v] += subtree_size[u];
68                 if (heavy_child[v] == -1 || subtree_size[u] >
69                     subtree_size[heavy_child[v]]) heavy_child[v] = u;
70             }
71         }
72
73     void decompose(int v, int chead) {
74         // start a new path
75         if (chead == -1) chead = v;
76
77         // consecutive ids in the hld path
78         at[cpos] = v;
79         pos[v] = cpos++;
80         head[v] = chead;
81
82         // if not a leaf
83         if (heavy_child[v] != -1) decompose(heavy_child[v],
84             chead);
85
86         // light child
87         for (auto u : adj[v]){
88             // start new path
89             if (u != parent[v] && u != heavy_child[v])
90                 decompose(u, -1);
91         }
92
93     ftype query_path(int a, int b) {
94         if(pos[a] < pos[b]) swap(a, b);
95         if(head[a] == head[b]) return seg.query(pos[b],
96             pos[a]);
97         return seg.f(seg.query(pos[head[a]], pos[a]),
98             query_path(parent[head[a]], b));
99     }
100
101     // iterative
102     /*ftype query_path(int a, int b) {
103         ftype ans = 0;
104
105         while (head[a] != head[b]) {
106             if (level[head[a]] > level[head[b]]) swap(a, b)
107             ;
108             ans = seg.merge(ans, seg.query(pos[head[b]],
109                 pos[b]));
110             b = parent[head[b]];
111         }
112
113         if (level[a] > level[b]) swap(a, b);
114         ans = seg.merge(ans, seg.query(pos[a], pos[b]));
115         return ans;
116     }*/
117
118     ftype query_subtree(int a) {
119         return seg.query(pos[a], pos[a] + subtree_size[a]
120             - 1);
121     }
122
123     void update_path(int a, int b, int x) {
124         if(pos[a] < pos[b]) swap(a, b);
125
126         if(head[a] == head[b]) return (void)seg.update(
127             pos[b], pos[a], x);
128         seg.update(pos[head[a]], pos[a], x); update_path(
129             parent[head[a]], b, x);
130     }
131
132     void update_subtree(int a, int val) {
133         seg.update(pos[a], pos[a] + subtree_size[a] - 1,
134             val);
135     }
136
137     void update(int a, int val) {
138         seg.update(pos[a], pos[a], val);
139     }
140
141     //edge
142     void update(int a, int b, int val) {
143         if (level[a] > level[b]) swap(a, b);
144         update(b, val);
145     }
146
147     int lca(int a, int b) {
148         if(pos[a] < pos[b]) swap(a, b);
149         return head[a] == head[b] ? b : lca(parent[head[a]
150             ], b);
151     }
152
153     void search(int a) {
154         a = parent[a];
155         if (a == -1) return;
156         if (seg.query(pos[head[a]], pos[head[a]]+
157             subtree_size[head[a]]-1) + pos[a]-pos[head[a]]+1
158             == subtree_size[head[a]]) {
159             seg.update(pos[head[a]], pos[a], 1);
160             return search(parent[head[a]]);
161         }
162         int l = pos[head[a]], r = pos[a]+1;
163         while (l < r) {
164             int m = (l+r)/2;
165             if (seg.query(m, m+subtree_size[at[m]]-1) + pos
166                 [a]-m+1 == subtree_size[at[m]]) {
167                 r = m;

```

```

156     }
157     else l = m+1;
158 }
159 seg.update(l, pos[a], 1);
160 }
161
162 /* k-th ancestor of x
163 int x, k; cin >> x >> k;
164
165 for (int b = 0; b <= BITS; b++) {
166     if (x != -1 && (k & (1 << b))) {
167         x = up[x][b];
168     }
169 }
170
171 cout << x << '\n';
172 */
173 void preprocess() {
174     up.assign(n + 1, vector<int>(31, -1));
175
176     for (int i = 1; i < n; i++) {
177         up[i][0] = parent[i];
178     }
179
180     for (int i = 1; i < n; i++) {
181         for (int j = 1; j <= 30; j++) {
182             if (up[i][j - 1] != -1) up[i][j] = up[up[i][j - 1]][j - 1];
183         }
184     }
185 }
186
187 int getKth(int p, int q, int k){
188     int a = lca(p,q), d;
189
190     if( a == p ){
191         d = level[q] - level[p] + 1;
192         swap(p,q);
193         k = d - k + 1;
194     }
195     else if( a == q );
196     else {
197         if( k > level[p] - level[a] + 1 ) {
198             d = level[p] + level[q] - 2 * level[a] +
199             1;
200             k = d - k + 1;
201             swap(p,q);
202         }
203         else ;
204     }
205     int lg; for( lg = 1; (1 << lg) <= level[p]; ++lg ); lg--;
206     k--;
207     for( int i = lg; i >= 0; i-- ){
208         if( (1 << i) <= k ){
209             p = up[p][i];
210             k -= (1 << i);
211         }
212     }
213     return p;
214 };

```

1.3 Hld Edge

```

1 // Description:
2 // Make queries and updates between two vertexes on a
   tree
3
4 // Problem:
5 // https://www.spoj.com/problems/QTREE/
6
7 // Complexity:

```

```

8 // O(log ^2 n) for both query and update
9
10 // How to use:
11 // HLD hld = HLD(n + 1, adj)
12
13 // Notes
14 // Change the root of the tree on the constructor if
   it's different from 1
15 // Use together with Segtree
16
17 struct HLD {
18     vector<int> parent;
19     vector<int> pos;
20     vector<int> head;
21     vector<int> subtree_size;
22     vector<int> level;
23     vector<int> heavy_child;
24     vector<ftype> subtree_weight;
25     vector<ftype> path_weight;
26     vector<vector<int>> adj;
27     vector<int> at;
28     Segtree seg = Segtree(0);
29     int cpos;
30     int n;
31     int root;
32
33     HLD() {}
34
35     HLD(int n, vector<vector<int>>& adj, int root = 1)
       : adj(adj), n(n), root(root) {
36         seg = Segtree(n);
37         cpos = 0;
38         at.assign(n, 0);
39         parent.assign(n, 0);
40         pos.assign(n, 0);
41         head.assign(n, 0);
42         subtree_size.assign(n, 1);
43         level.assign(n, 0);
44         heavy_child.assign(n, -1);
45         parent[root] = -1;
46         dfs(root, -1);
47         decompose(root, -1);
48     }
49
50     void dfs(int v, int p) {
51         parent[v] = p;
52         if (p != -1) level[v] = level[p] + 1;
53         for (auto u : adj[v]) {
54             if (u != p) {
55                 dfs(u, v);
56                 subtree_size[v] += subtree_size[u];
57                 if (heavy_child[v] == -1 || subtree_size[u] >
                   subtree_size[heavy_child[v]]) heavy_child[v] = u;
58             }
59         }
60     }
61
62     void decompose(int v, int chead) {
63         // start a new path
64         if (chead == -1) chead = v;
65
66         // consecutive ids in the hld path
67         at[cpos] = v;
68         pos[v] = cpos++;
69         head[v] = chead;
70
71         // if not a leaf
72         if (heavy_child[v] != -1) decompose(heavy_child[v], chead);
73
74         // light child
75         for (auto u : adj[v]){

```

```

76     // start new path
77     if (u != parent[v] && u != heavy_child[v])
78         decompose(u, -1);
79 }
80
81 ll query_path(int a, int b) {
82     if (a == b) return 0;
83     if(pos[a] < pos[b]) swap(a, b);
84
85     if(head[a] == head[b]) return seg.query(pos[b] +
86     1, pos[a]);
87     return seg.f(seg.query(pos[head[a]], pos[a]),
88     query_path(parent[head[a]], b));
89 }
90
91 ftype query_subtree(int a) {
92     if (subtree_size[a] == 1) return 0;
93     return seg.query(pos[a] + 1, pos[a] +
94     subtree_size[a] - 1);
95 }
96
97 void update_path(int a, int b, int x) {
98     if (a == b) return;
99     if(pos[a] < pos[b]) swap(a, b);
100
101     if(head[a] == head[b]) return (void)seg.update(
102     pos[b] + 1, pos[a], x);
103     seg.update(pos[head[a]], pos[a], x); update_path(
104     parent[head[a]], b, x);
105 }
106
107 void update_subtree(int a, int val) {
108     if (subtree_size[a] == 1) return;
109     seg.update(pos[a] + 1, pos[a] + subtree_size[a] -
110     1, val);
111 }
112
113 // vertex
114 void update(int a, int val) {
115     seg.update(pos[a], pos[a], val);
116 }
117
118 //edge
119 void update(int a, int b, int val) {
120     if (parent[a] == b) swap(a, b);
121     update(b, val);
122 }
123
124 int lca(int a, int b) {
125     if(pos[a] < pos[b]) swap(a, b);
126     return head[a] == head[b] ? b : lca(parent[head[a]],
127     b);
128 }
129 };

```

1.4 Tarjan Bridge

```

1 // Description:
2 // Find a bridge in a connected undirected graph
3 // A bridge is an edge so that if you remove that
4 // edge the graph is no longer connected
5
6 // Problem:
7 // https://cses.fi/problemset/task/2177/
8
9 // Complexity:
10 // O(V + E) where V is the number of vertices and E
11 // is the number of edges
12
13 int n;
14 vector<vector<int>> adj;

```

```

14 vector<bool> visited;
15 vector<int> tin, low;
16 int timer;
17
18 void dfs(int v, int p) {
19     visited[v] = true;
20     tin[v] = low[v] = timer++;
21     for (int to : adj[v]) {
22         if (to == p) continue;
23         if (visited[to]) {
24             low[v] = min(low[v], tin[to]);
25         } else {
26             dfs(to, v);
27             low[v] = min(low[v], low[to]);
28             if (low[to] > tin[v]) {
29                 IS_BRIDGE(v, to);
30             }
31         }
32     }
33 }
34
35 void find_bridges() {
36     timer = 0;
37     visited.assign(n, false);
38     tin.assign(n, -1);
39     low.assign(n, -1);
40     for (int i = 0; i < n; ++i) {
41         if (!visited[i])
42             dfs(i, -1);
43     }
44 }

```

1.5 2sat

```

1 // Description:
2 // Solves expression of the type (a v b) ^ (c v d) ^
3 // (e v f)
4
5 // Problem:
6 // https://cses.fi/problemset/task/1684
7
8 // Complexity:
9 // O(n + m) where n is the number of variables and m
10 // is the number of clauses
11
12 #include <bits/stdc++.h>
13 #define pb push_back
14 #define mp make_pair
15 #define pii pair<int, int>
16 #define ff first
17 #define ss second
18
19 using namespace std;
20
21 struct SAT {
22     int nodes;
23     int curr = 0;
24     int component = 0;
25     vector<vector<int>> adj;
26     vector<vector<int>> rev;
27     vector<vector<int>> condensed;
28     vector<pii> departure;
29     vector<bool> visited;
30     vector<int> scc;
31     vector<int> order;
32
33     // 1 to nodes
34     // nodes + 1 to 2 * nodes
35     SAT(int nodes) : nodes(nodes) {
36         adj.resize(2 * nodes + 1);
37         rev.resize(2 * nodes + 1);
38         visited.resize(2 * nodes + 1);
39         scc.resize(2 * nodes + 1);

```

```

38     }
39
40     void add_imp(int a, int b) {
41         adj[a].pb(b);
42         rev[b].pb(a);
43     }
44
45     int get_not(int a) {
46         if (a > nodes) return a - nodes;
47         return a + nodes;
48     }
49
50     void add_or(int a, int b) {
51         add_imp(get_not(a), b);
52         add_imp(get_not(b), a);
53     }
54
55     void add_nor(int a, int b) {
56         add_or(get_not(a), get_not(b));
57     }
58
59     void add_and(int a, int b) {
60         add_or(get_not(a), b);
61         add_or(a, get_not(b));
62         add_or(a, b);
63     }
64
65     void add_nand(int a, int b) {
66         add_or(get_not(a), b);
67         add_or(a, get_not(b));
68         add_or(get_not(a), get_not(b));
69     }
70
71     void add_xor(int a, int b) {
72         add_or(a, b);
73         add_or(get_not(a), get_not(b));
74     }
75
76     void add_xnor(int a, int b) {
77         add_or(get_not(a), b);
78         add_or(a, get_not(b));
79     }
80
81     void departure_time(int v) {
82         visited[v] = true;
83
84         for (auto u : adj[v]) {
85             if (!visited[u]) departure_time(u);
86         }
87
88         departure.pb(mp(++curr, v));
89     }
90
91     void find_component(int v, int component) {
92         scc[v] = component;
93         visited[v] = true;
94
95         for (auto u : rev[v]) {
96             if (!visited[u]) find_component(u,
97             component);
98         }
99     }
100
101     void topological_order(int v) {
102         visited[v] = true;
103
104         for (auto u : condensed[v]) {
105             if (!visited[u]) topological_order(u);
106         }
107
108         order.pb(v);
109     }

```

```

110     bool is_possible() {
111         component = 0;
112         for (int i = 1; i <= 2 * nodes; i++) {
113             if (!visited[i]) departure_time(i);
114         }
115
116         sort(departure.begin(), departure.end(),
117             greater<pii>());
118
119         visited.assign(2 * nodes + 1, false);
120
121         for (auto [_ , node] : departure) {
122             if (!visited[node]) find_component(node,
123             ++component);
124         }
125
126         for (int i = 1; i <= nodes; i++) {
127             if (scc[i] == scc[i + nodes]) return
128             false;
129         }
130
131         return true;
132     }
133
134     int find_value(int e, vector<int> &ans) {
135         if (e > nodes && ans[e - nodes] != 2) return
136         !ans[e - nodes];
137         if (e <= nodes && ans[e + nodes] != 2) return
138         !ans[e + nodes];
139         return 0;
140     }
141
142     vector<int> find_ans() {
143         condensed.resize(component + 1);
144
145         for (int i = 1; i <= 2 * nodes; i++) {
146             for (auto u : adj[i]) {
147                 if (scc[i] != scc[u]) condensed[scc[i]
148                 ].pb(scc[u]);
149             }
150         }
151
152         visited.assign(component + 1, false);
153
154         for (int i = 1; i <= component; i++) {
155             if (!visited[i]) topological_order(i);
156         }
157
158         reverse(order.begin(), order.end());
159
160         // 0 - false
161         // 1 - true
162         // 2 - no value yet
163         vector<int> ans(2 * nodes + 1, 2);
164
165         vector<vector<int>> belong(component + 1);
166
167         for (int i = 1; i <= 2 * nodes; i++) {
168             belong[scc[i]].pb(i);
169         }
170
171         for (auto p : order) {
172             for (auto e : belong[p]) {
173                 ans[e] = find_value(e, ans);
174             }
175         }
176
177         return ans;
178     }
179 };
180
181 int main() {
182     ios::sync_with_stdio(false);

```

```

177     cin.tie(NULL);
178
179     int n, m; cin >> n >> m;
180
181     SAT sat = SAT(m);
182
183     for (int i = 0; i < n; i++) {
184         char op1, op2; int a, b; cin >> op1 >> a >>
185         op2 >> b;
186         if (op1 == '+' && op2 == '+') sat.add_or(a, b);
187         if (op1 == '-' && op2 == '-') sat.add_or(sat.get_not(a), sat.get_not(b));
188         if (op1 == '+' && op2 == '-') sat.add_or(a, sat.get_not(b));
189         if (op1 == '-' && op2 == '+') sat.add_or(sat.get_not(a), b);
190     }
191
192     if (!sat.is_possible()) cout << "IMPOSSIBLE\n";
193     else {
194         vector<int> ans = sat.find_ans();
195         for (int i = 1; i <= m; i++) {
196             cout << (ans[i] == 1 ? '+' : '-') << ' ';
197         }
198         cout << '\n';
199     }
200     return 0;
201 }

```

1.6 Dijkstra

```

1  const int MAX = 2e5+7;
2  const int INF = 1000000000;
3  vector<vector<pair<int, int>>> adj(MAX);
4
5  void dijkstra(int s, vector<int> & d, vector<int> & p)
6  {
7      int n = adj.size();
8      d.assign(n, INF);
9      p.assign(n, -1);
10
11      d[s] = 0;
12      set<pair<int, int>> q;
13      q.insert({0, s});
14      while (!q.empty()) {
15          int v = q.begin()->second;
16          q.erase(q.begin());
17
18          for (auto edge : adj[v]) {
19              int to = edge.first;
20              int len = edge.second;
21
22              if (d[v] + len < d[to]) {
23                  q.erase({d[to], to});
24                  d[to] = d[v] + len;
25                  p[to] = v;
26                  q.insert({d[to], to});
27              }
28          }
29      }
30
31      vector<int> restore_path(int s, int t) {
32          vector<int> path;
33
34          for (int v = t; v != s; v = p[v])
35              path.push_back(v);
36          path.push_back(s);
37
38          reverse(path.begin(), path.end());
39          return path;

```

```

40 }
41
42 int adj[MAX][MAX];
43 int dist[MAX];
44 int minDistance(int dist[], bool sptSet[], int V) {
45     int min = INT_MAX, min_index;
46
47     for (int v = 0; v < V; v++)
48         if (sptSet[v] == false && dist[v] <= min)
49             min = dist[v], min_index = v;
50
51     return min_index;
52 }
53
54 void dijkstra(int src, int V) {
55     bool sptSet[V];
56     for (int i = 0; i < V; i++)
57         dist[i] = INT_MAX, sptSet[i] = false;
58
59     dist[src] = 0;
60
61     for (int count = 0; count < V - 1; count++) {
62         int u = minDistance(dist, sptSet, V);
63
64         sptSet[u] = true;
65
66         for (int v = 0; v < V; v++)
67             if (!sptSet[v] && adj[u][v]
68                 && dist[u] != INT_MAX
69                 && dist[u] + adj[u][v] < dist[v])
70                 dist[v] = dist[u] + adj[u][v];
71     }
72 }
73
74 }

```

1.7 Ford Fulkerson Edmonds Karp

```

1 // Description:
2 // Obtains the maximum possible flow rate given a
3 // network. A network is a graph with a single
4 // source vertex and a single sink vertex in which
5 // each edge has a capacity
6
7 // Complexity:
8 //  $O(V * E^2)$  where V is the number of vertex and E
9 // is the number of edges
10
11 int n;
12 vector<vector<int>> capacity;
13 vector<vector<int>> adj;
14
15 int bfs(int s, int t, vector<int> & parent) {
16     fill(parent.begin(), parent.end(), -1);
17     parent[s] = -2;
18     queue<pair<int, int>> q;
19     q.push({s, INF});
20
21     while (!q.empty()) {
22         int cur = q.front().first;
23         int flow = q.front().second;
24         q.pop();
25
26         for (int next : adj[cur]) {
27             if (parent[next] == -1 && capacity[cur][
28                 next]) {
29                 parent[next] = cur;
30                 int new_flow = min(flow, capacity[cur
31                     ][next]);
32                 if (next == t)
33                     return new_flow;
34                 q.push({next, new_flow});
35             }
36         }
37     }
38 }

```



```

30     }
31 }
32
33 return 0;
34 }
35
36 int maxflow(int s, int t) {
37     int flow = 0;
38     vector<int> parent(n);
39     int new_flow;
40
41     while (new_flow = bfs(s, t, parent)) {
42         flow += new_flow;
43         int cur = t;
44         while (cur != s) {
45             int prev = parent[cur];
46             capacity[prev][cur] -= new_flow;
47             capacity[cur][prev] += new_flow;
48             cur = prev;
49         }
50     }
51
52     return flow;
53 }

```

1.8 Bipartite

```

1  const int NONE = 0, BLUE = 1, RED = 2;
2  vector<vector<int>> graph(100005);
3  vector<bool> visited(100005);
4  int color[100005];
5
6  bool bfs(int s = 1){
7
8      queue<int> q;
9      q.push(s);
10     color[s] = BLUE;
11
12     while (not q.empty()){
13         auto u = q.front(); q.pop();
14
15         for (auto v : graph[u]){
16             if (color[v] == NONE){
17                 color[v] = 3 - color[u];
18                 q.push(v);
19             }
20             else if (color[v] == color[u]){
21                 return false;
22             }
23         }
24     }
25
26     return true;
27 }
28
29 bool is_bipartite(int n){
30
31     for (int i = 1; i<=n; i++)
32         if (color[i] == NONE and not bfs(i))
33             return false;
34
35     return true;
36 }

```

1.9 Floyd Warshall

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  using ll = long long;
5
6  const int MAX = 507;

```

```

7  const long long INF = 0x3f3f3f3f3f3f3fLL;
8
9  ll dist[MAX][MAX];
10 int n;
11
12 void floyd_warshall() {
13     for (int i = 0; i < n; i++) {
14         for (int j = 0; j < n; j++) {
15             if (i == j) dist[i][j] = 0;
16             else if (!dist[i][j]) dist[i][j] = INF;
17         }
18     }
19
20     for (int k = 0; k < n; k++) {
21         for (int i = 0; i < n; i++) {
22             for (int j = 0; j < n; j++) {
23                 // trata o caso no qual o grafo tem
24                 // arestas com peso negativo
25                 if (dist[i][k] < INF && dist[k][j] <
26                     INF){
27                     dist[i][j] = min(dist[i][j], dist
28                                     [i][k] + dist[k][j]);
29                 }
30             }
31         }
32     }
33 }

```

1.10 Hungarian

```

1  // Description:
2  // A matching algorithm for weighted bipartite graphs
3  // that returns
4  // a perfect match
5
6  // Problem:
7  // https://codeforces.com/gym/103640/problem/H
8
9  // Complexity:
10 // O(V ^ 3) in which V is the number of vertexs
11
12 // Notes:
13 // Indexed at 1
14 // n is the number of items on the right side and m
15 // the number of items
16 // on the left side of the graph
17 // Returns minimum assignment cost and which items
18 // were matched
19
20 pair<int, vector<pii>> hungarian(int n, int m, vector
21 <vector<int>> A) {
22     vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
23     for (int i=1; i<=n; ++i) {
24         p[0] = i;
25         int j0 = 0;
26         vector<int> minv (m+1, INF);
27         vector<char> used (m+1, false);
28         do {
29             used[j0] = true;
30             int i0 = p[j0], delta = INF, j1;
31             for (int j=1; j<=m; ++j)
32                 if (!used[j]) {
33                     int cur = A[i0][j]-u[i0]-v[j];
34                     if (cur < minv[j])
35                         minv[j] = cur, way[j] = j0;
36                     if (minv[j] < delta)
37                         delta = minv[j], j1 = j;
38                 }
39             for (int j=0; j<=m; ++j)
40                 if (used[j])
41                     u[p[j]] += delta, v[j] -= delta;

```

```

40         else
41             minv[j] -= delta;
42         j0 = j1;
43     } while (p[j0] != 0);
44     do {
45         int j1 = way[j0];
46         p[j0] = p[j1];
47         j0 = j1;
48     } while (j0);
49 }
50
51 vector<pair<int, int>> result;
52 for (int i = 1; i <= m; ++i){
53     result.push_back(make_pair(p[i], i));
54 }
55
56 int C = -v[0];
57
58 return mp(C, result);
59 }

```

1.11 Centroid Decomposition

```

1  int n;
2  vector<set<int>> adj;
3  vector<char> ans;
4
5  vector<bool> removed;
6
7  vector<int> subtree_size;
8
9  int dfs(int u, int p = 0) {
10     subtree_size[u] = 1;
11
12     for(int v : adj[u]) {
13         if(v != p && !removed[v]) {
14             subtree_size[u] += dfs(v, u);
15         }
16     }
17
18     return subtree_size[u];
19 }
20
21 int get_centroid(int u, int sz, int p = 0) {
22     for(int v : adj[u]) {
23         if(v != p && !removed[v]) {
24             if(subtree_size[v]*2 > sz) {
25                 return get_centroid(v, sz, u);
26             }
27         }
28     }
29
30     return u;
31 }
32
33 char get_next(char c) {
34     if (c != 'Z') return c + 1;
35     return '$';
36 }
37
38 bool flag = true;
39
40 void solve(int node, char c) {
41     int center = get_centroid(node, dfs(node));
42     ans[center] = c;
43     removed[center] = true;
44
45     for (auto u : adj[center]) {
46         if (!removed[u]) {
47             char next = get_next(c);
48             if (next == '$') {
49                 flag = false;
50                 return;

```

```

51     }
52     solve(u, next);
53 }
54 }
55 }
56
57 int32_t main(){
58     ios::sync_with_stdio(false);
59     cin.tie(NULL);
60
61     cin >> n;
62     adj.resize(n + 1);
63     ans.resize(n + 1);
64     removed.resize(n + 1);
65     subtree_size.resize(n + 1);
66
67     for (int i = 1; i <= n - 1; i++) {
68         int u, v; cin >> u >> v;
69         adj[u].insert(v);
70         adj[v].insert(u);
71     }
72
73     solve(1, 'A');
74
75     if (!flag) cout << "Impossible!\n";
76     else {
77         for (int i = 1; i <= n; i++) {
78             cout << ans[i] << ' ';
79         }
80         cout << '\n';
81     }
82
83     return 0;
84 }

```

1.12 Tree Diameter

```

1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  const int MAX = 3e5+17;
6
7  vector<int> adj[MAX];
8  bool visited[MAX];
9
10 int max_depth = 0, max_node = 1;
11
12 void dfs (int v, int depth) {
13     visited[v] = true;
14
15     if (depth > max_depth) {
16         max_depth = depth;
17         max_node = v;
18     }
19
20     for (auto u : adj[v]) {
21         if (!visited[u]) dfs(u, depth + 1);
22     }
23 }
24
25 int tree_diameter() {
26     dfs(1, 0);
27     max_depth = 0;
28     for (int i = 0; i < MAX; i++) visited[i] = false;
29     dfs(max_node, 0);
30     return max_depth;
31 }

```

1.13 Kuhn

```

1 // Description

```

```

2 // Matching algorithm for unweighted bipartite graph
3 ::
4 // Problem:
5 // https://codeforces.com/gym/104252/problem/H
6
7 // Complexity:
8 //  $O(V * E)$  in which  $V$  is the number of vertexes and
9 //  $E$  is the number of edges
10
11 // Notes:
12 // Indexed at zero
13
14 int n, k;
15 // adjacency list
16 vector<vector<int>> g;
17 vector<int> mt;
18 vector<bool> used;
19
20 bool try_kuhn(int v) {
21     if (used[v])
22         return false;
23     used[v] = true;
24     for (int to : g[v]) {
25         if (mt[to] == -1 || try_kuhn(mt[to])) {
26             mt[to] = v;
27             return true;
28         }
29     }
30     return false;
31 }
32
33 int main() {
34     // ... reading the graph g ...
35
36     mt.assign(k, -1);
37     vector<bool> used1(n, false);
38     for (int v = 0; v < n; ++v) {
39         for (int to : g[v]) {
40             if (mt[to] == -1) {
41                 mt[to] = v;
42                 used1[v] = true;
43                 break;
44             }
45         }
46     }
47     for (int v = 0; v < n; ++v) {
48         if (used1[v])
49             continue;
50         used.assign(n, false);
51         try_kuhn(v);
52     }
53
54     for (int i = 0; i < k; ++i)
55         if (mt[i] != -1)
56             printf("%d %d\n", mt[i] + 1, i + 1);
57 }

```

1.14 Negative Cycle

```

1 // Description
2 // Detects any cycle in which the sum of edge weights
3 // is negative.
4 // Alternatively, we can detect whether there is a
5 // negative cycle
6 // starting from a specific vertex.
7
8 // Problem:
9 // https://cses.fi/problemset/task/1197
10
11 // Complexity:
12 //  $O(n * m)$ 
13

```

```

14 // Notes
15 // In order to consider only the negative cycles
16 // located on the path from a to b,
17 // Reverse the graph, run a dfs from node b and mark
18 // the visited nodes
19 // Consider only the edges that connect to visited
20 // nodes when running bellman-ford
21 // on the normal graph
22
23 struct Edge {
24     int a, b, cost;
25     Edge(int a, int b, int cost) : a(a), b(b), cost(
26         cost) {}
27 };
28
29 int n, m;
30 vector<Edge> edges;
31 const int INF = 1e9+10;
32
33 void negative_cycle() {
34     // uncomment to find negative cycle starting from a
35     // vertex v
36     // vector<int> d(n + 1, INF);
37     // d[v] = 0;
38     vector<int> d(n + 1, 0);
39     vector<int> p(n + 1, -1);
40     int x;
41     // uncomment to find all negative cycles
42     // // set<int> s;
43     for (int i = 1; i <= n; ++i) {
44         x = -1;
45         for (Edge e : edges) {
46             // if (d[e.a] >= INF) continue;
47             if (d[e.b] > d[e.a] + e.cost) {
48                 // d[e.b] = max(-INF, d[e.a] + e.cost);
49                 d[e.b] = d[e.a] + e.cost;
50                 p[e.b] = e.a;
51                 x = e.b;
52                 // // s.insert(e.b);
53             }
54         }
55     }
56
57     if (x == -1)
58         cout << "NO\n";
59     else {
60         // // int y = all nodes in set s
61         int y = x;
62         for (int i = 1; i <= n; ++i) {
63             y = p[y];
64         }
65
66         vector<int> path;
67         for (int cur = y; cur = p[cur]) {
68             path.push_back(cur);
69             if (cur == y && path.size() > 1) break;
70         }
71         reverse(path.begin(), path.end());
72
73         cout << "YES\n";
74         for (int u : path)
75             cout << u << ' ';
76         cout << '\n';
77     }
78 }

```

1.15 Eulerian Undirected

```

1 // Description:
2 // Hierholzer's Algorithm
3 // An Eulerian path is a path that passes through
4 // every edge exactly once.

```

```

4 // An Eulerian circuit is an Eulerian path that
  starts and ends on the same node.
5
6 // An Eulerian path exists in an undirected graph if
  the degree of every node is even (not counting
  self-edges)
7 // except for possibly exactly two nodes that have
  and odd degree (start and end nodes).
8 // An Eulerian circuit exists in an undirected graph
  if the degree of every node is even.
9
10 // The graph has to be connected (except for isolated
   nodes which are allowed because there
11 // are no edges connected to them).
12
13 // Problem:
14 // https://cses.fi/problemset/task/1691
15
16 // Complexity:
17 //  $O(E * \log(E))$  where E is the number of edges
18
19 // How to use
20 // Check whether the path exists before trying to
   find it
21 // Find the root - any node that has at least 1
   outgoing edge
22 // (if the problem requires that you start from a
   node v, the root will be the node v)
23 // Count the degree;
24 //
25 // for (int i = 0; i < m; i++) {
26 //   int a, b; cin >> a >> b;
27 //   adj[a].pb(b); adj[b].pb(a);
28 //   root = a;
29 //   degree[a]++; degree[b]++;
30 // }
31
32 // Notes
33 // If you want to find a path start and ending nodes
   v and u
34 // if ((is_eulerian(n, root, start, end) != 1) || (
   start != v) || (end != u)) cout << "IMPOSSIBLE\n"
35
36 // It can be speed up to work on  $O(E)$  on average by
   using unordered_set instead of set
37
38 // It works when there are self loops, but not when
   there are multiple edges
39 // It the graph has multiple edges, add more notes to
   simulate the edges
40 // e.g
41 // 1 2
42 // 1 2
43 // 1 2
44 // becomes
45 // 3 4
46 // 4 1
47 // 1 2
48
49 vector<bool> visited;
50 vector<int> degree;
51 vector<vector<int>> adj;
52
53 void dfs(int v) {
54     visited[v] = true;
55     for (auto u : adj[v]) {
56         if (!visited[u]) dfs(u);
57     }
58 }
59
60 int is_eulerian(int n, int root, int& start, int& end
  ) {
61     start = -1, end = -1;
62     if (n == 1) return 2; // only one node
63     visited.assign(n + 1, false);
64     dfs(root);
65
66     for (int i = 1; i <= n; i++) {
67         if (!visited[i] && degree[i] > 0) return 0;
68     }
69
70     for (int i = 1; i <= n; i++) {
71         if (start == -1 && degree[i] % 2 == 1) start = i;
72         else if (end == -1 && degree[i] % 2 == 1) end = i
73         ;
74         else if (degree[i] % 2 == 1) return 0;
75     }
76
77     if (start == -1 && end == -1) {start = root; end =
78         root; return 2;} // has eulerian circuit and path
79     if (start != -1 && end != -1) return 1; // has
80         eulerian path
81     return 0; // no eulerian path nor circuit
82 }
83
84 vector<int> path;
85 vector<set<int>> mark;
86
87 void dfs_path(int v) {
88     visited[v] = true;
89
90     while (degree[v] != 0) {
91         degree[v]--;
92         int u = adj[v][degree[v]];
93         if (mark[v].find(u) != mark[v].end()) continue;
94         mark[v].insert(u);
95         mark[u].insert(v);
96         int next_edge = adj[v][degree[v]];
97         dfs_path(next_edge);
98     }
99     path.pb(v);
100 }
101
102 void find_path(int n, int start) {
103     path.clear();
104     mark.resize(n + 1);
105     visited.assign(n + 1, false);
106     dfs_path(start);
107 }

```

1.16 Bellman Ford

```

1 // Description:
2 // Finds the shortest path from a vertex v to any
   other vertex
3
4 // Problem:
5 // https://cses.fi/problemset/task/1673
6
7 // Complexity:
8 //  $O(n * m)$ 
9
10 struct Edge {
11     int a, b, cost;
12     Edge(int a, int b, int cost) : a(a), b(b), cost(
13         cost) {}
14 };
15
16 int n, m;
17 vector<Edge> edges;
18 const int INF = 1e9+10;
19
20 void bellman_ford(int v, int t) {
21     vector<int> d(n + 1, INF);
22     d[v] = 0;
23     vector<int> p(n + 1, -1);

```

```

23
24 for (;;) {
25     bool any = false;
26     for (Edge e : edges) {
27         if (d[e.a] >= INF) continue;
28         if (d[e.b] > d[e.a] + e.cost) {
29             d[e.b] = d[e.a] + e.cost;
30             p[e.b] = e.a;
31             any = true;
32         }
33     }
34     if (!any) break;
35 }
36
37 if (d[t] == INF)
38     cout << "No path from " << v << " to " << t << ".
39 ";
40 else {
41     vector<int> path;
42     for (int cur = t; cur != -1; cur = p[cur]) {
43         path.push_back(cur);
44     }
45     reverse(path.begin(), path.end());
46     cout << "Path from " << v << " to " << t << ": ";
47     for (int u : path) {
48         cout << u << ' ';
49     }
50 }
51 }

```

1.17 Blossom

```

1 // Description:
2 // Matching algorithm for general graphs (non-
3 // bipartite)
4
5 // Problem:
6 // https://acm.timus.ru/problem.aspx?space=1&num=1099
7
8 // Complexity:
9 // O(n^3)
10
11 // vector<pii> Blossom(vector<vector<int>>& graph) {
12 vector<int> Blossom(vector<vector<int>>& graph) {
13     int n = graph.size(), timer = -1;
14     vector<int> mate(n, -1), label(n), parent(n),
15         orig(n), aux(n, -1), q;
16     auto lca = [&](int x, int y) {
17         for (timer++; ; swap(x, y)) {
18             if (x == -1) continue;
19             if (aux[x] == timer) return x;
20             aux[x] = timer;
21             x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
22         }
23     };
24     auto blossom = [&](int v, int w, int a) {
25         while (orig[v] != a) {
26             parent[v] = w; w = mate[v];
27             if (label[w] == 1) label[w] = 0, q.push_back(w);
28         };
29         orig[v] = orig[w] = a; v = parent[w];
30     };
31     auto augment = [&](int v) {
32         while (v != -1) {
33             int pv = parent[v], nv = mate[pv];
34             mate[v] = pv; mate[pv] = v; v = nv;
35         }
36     };
37     auto bfs = [&](int root) {
38         fill(label.begin(), label.end(), -1);

```

```

38     iota(orig.begin(), orig.end(), 0);
39     q.clear();
40     label[root] = 0; q.push_back(root);
41     for (int i = 0; i < (int)q.size(); ++i) {
42         int v = q[i];
43         for (auto x : graph[v]) {
44             if (label[x] == -1) {
45                 label[x] = 1; parent[x] = v;
46                 if (mate[x] == -1)
47                     return augment(x), 1;
48                 label[mate[x]] = 0; q.push_back(mate[x]);
49             } else if (label[x] == 0 && orig[v] != orig[x])
50                 ) {
51                     int a = lca(orig[v], orig[x]);
52                     blossom(x, v, a); blossom(v, x, a);
53                 }
54             }
55         return 0;
56     };
57     // Time halves if you start with (any) maximal
58     // matching.
59     for (int i = 0; i < n; i++)
60         if (mate[i] == -1)
61             bfs(i);
62     return mate;
63
64     /*
65     vector<bool> used(n, false);
66     vector<pii> ans;
67     for (int i = 0; i < n; i++) {
68         if (matching[i] == -1 || used[i]) continue;
69         used[i] = true;
70         used[matching[i]] = true;
71         ans.emplace_back(i, matching[i]);
72     }
73     return ans;
74     */

```

1.18 Kruskal

```

1 struct DSU {
2     int n;
3     vector<int> link, sizes;
4
5     DSU(int n) {
6         this->n = n;
7         link.assign(n+1, 0);
8         sizes.assign(n+1, 1);
9
10         for (int i = 0; i <= n; i++)
11             link[i] = i;
12     }
13
14     int find(int x) {
15         while (x != link[x])
16             x = link[x];
17
18         return x;
19     }
20
21     bool same(int a, int b) {
22         return find(a) == find(b);
23     }
24
25     void unite(int a, int b) {
26         a = find(a);
27         b = find(b);
28
29         if (a == b) return;
30
31         if (sizes[a] < sizes[b])

```

```

32         swap(a, b);
33
34         sizes[a] += sizes[b];
35         link[b] = a;
36     }
37 };
38
39 struct Edge {
40     int u, v;
41     long long weight;
42
43     Edge() {}
44
45     Edge(int u, int v, long long weight) : u(u), v(v)
46     , weight(weight) {}
47
48     bool operator<(const Edge& other) const {
49         return weight < other.weight;
50     }
51
52     bool operator>(const Edge& other) const {
53         return weight > other.weight;
54     }
55 };
56
57 vector<Edge> kruskal(vector<Edge> edges, int n) {
58     vector<Edge> result; // arestas da MST
59     long long cost = 0;
60
61     sort(edges.begin(), edges.end());
62
63     DSU dsu(n);
64
65     for (auto e : edges) {
66         if (!dsu.same(e.u, e.v)) {
67             cost += e.weight;
68             result.push_back(e);
69             dsu.unite(e.u, e.v);
70         }
71     }
72
73     return result;
74 }

```

1.19 Small To Large

```

1 // Problem:
2 // https://codeforces.com/contest/600/problem/E
3
4 void process_colors(int curr, int parent) {
5
6     for (int n : adj[curr]) {
7         if (n != parent) {
8             process_colors(n, curr);
9
10            if (colors[curr].size() < colors[n].size
11                ()) {
12                sum_num[curr] = sum_num[n];
13                vmax[curr] = vmax[n];
14                swap(colors[curr], colors[n]);
15            }
16
17            for (auto [item, vzs] : colors[n]) {
18                if (colors[curr][item] + vzs > vmax[curr]
19                    ) {
20                    vmax[curr] = colors[curr][item] +
21                        vzs;
22                    sum_num[curr] = item;
23                }
24                else if (colors[curr][item] + vzs ==
25                    sum_num[curr]) {
26                    sum_num[curr] += item;
27                }
28            }
29        }
30    }
31 }

```

```

24
25         colors[curr][item] += vzs;
26     }
27 }
28 }
29
30 }
31
32
33 int32_t main() {
34
35     int n; cin >> n;
36
37     for (int i = 1; i <= n; i++) {
38         int a; cin >> a;
39         colors[i][a] = 1;
40         vmax[i] = 1;
41         sum_num[i] = a;
42     }
43
44     for (int i = 1; i < n; i++) {
45         int a, b; cin >> a >> b;
46
47         adj[a].push_back(b);
48         adj[b].push_back(a);
49     }
50
51     process_colors(1, 0);
52
53     for (int i = 1; i <= n; i++) {
54         cout << sum_num[i] << (i < n ? " " : "\n");
55     }
56
57     return 0;
58 }
59
60

```

1.20 Prim

```

1 int n;
2 vector<vector<int>>> adj; // adjacency matrix of graph
3 const int INF = 1000000000; // weight INF means there
4     is no edge
5
6 struct Edge {
7     int w = INF, to = -1;
8 };
9
10 void prim() {
11     int total_weight = 0;
12     vector<bool> selected(n, false);
13     vector<Edge> min_e(n);
14     min_e[0].w = 0;
15
16     for (int i=0; i<n; ++i) {
17         int v = -1;
18         for (int j = 0; j < n; ++j) {
19             if (!selected[j] && (v == -1 || min_e[j].
20                 w < min_e[v].w)) {
21                 v = j;
22             }
23
24             if (min_e[v].w == INF) {
25                 cout << "No MST!" << endl;
26                 exit(0);
27             }
28
29             selected[v] = true;
30             total_weight += min_e[v].w;
31             if (min_e[v].to != -1) {
32                 cout << v << " " << min_e[v].to << endl;
33             }
34         }
35     }
36 }

```

```

32     for (int to = 0; to < n; ++to) {
33         if (adj[v][to] < min_e[to].w)
34             min_e[to] = {adj[v][to], v};
35     }
36 }
37
38 cout << total_weight << endl;
39 }

```

1.21 Cycle Path Recovery

```

1  int n;
2  vector<vector<int>> adj;
3  vector<char> color;
4  vector<int> parent;
5  int cycle_start, cycle_end;
6
7  bool dfs(int v) {
8      color[v] = 1;
9      for (int u : adj[v]) {
10         if (color[u] == 0) {
11             parent[u] = v;
12             if (dfs(u))
13                 return true;
14         } else if (color[u] == 1) {
15             cycle_end = v;
16             cycle_start = u;
17             return true;
18         }
19     }
20     color[v] = 2;
21     return false;
22 }
23
24 void find_cycle() {
25     color.assign(n, 0);
26     parent.assign(n, -1);
27     cycle_start = -1;
28
29     for (int v = 0; v < n; v++) {
30         if (color[v] == 0 && dfs(v))
31             break;
32     }
33
34     if (cycle_start == -1) {
35         cout << "Acyclic" << endl;
36     } else {
37         vector<int> cycle;
38         cycle.push_back(cycle_start);
39         for (int v = cycle_end; v != cycle_start; v =
40             parent[v])
41             cycle.push_back(v);
42         cycle.push_back(cycle_start);
43         reverse(cycle.begin(), cycle.end());
44
45         cout << "Cycle found: ";
46         for (int v : cycle)
47             cout << v << " ";
48         cout << endl;
49     }
50 }

```

1.22 Min Cost Max Flow

```

1  // Dinitz Min Cost
2  const int INF = 0x3f3f3f3f3f3f3f3f;
3
4  struct Dinitz {
5      struct Edge {
6          int v, u, cap, flow=0, cost;
7          Edge(int v, int u, int cap, int cost) : v(v), u(u
8              ), cap(cap), cost(cost) {}

```

```

8      };
9
10     int n, s, t;
11     Dinitz(int n, int s, int t) : n(n), s(s), t(t) {
12         adj.resize(n);
13     }
14
15     vector<Edge> edges;
16     vector<vector<int>> adj;
17     void add_edge(int v, int u, int cap, int cost) {
18         edges.emplace_back(v, u, cap, cost);
19         adj[v].push_back(size(edges)-1);
20         edges.emplace_back(u, v, 0, -cost);
21         adj[u].push_back(size(edges)-1);
22     }
23
24     vector<int> dist;
25     bool spfa() {
26         dist.assign(n, INF);
27
28         queue<int> Q;
29         vector<bool> inqueue(n, false);
30
31         dist[s] = 0;
32         Q.push(s);
33         inqueue[s] = true;
34
35         vector<int> cnt(n);
36
37         while (!Q.empty()) {
38             int v = Q.front(); Q.pop();
39             inqueue[v] = false;
40
41             for (auto eid : adj[v]) {
42                 auto const& e = edges[eid];
43                 if (e.cap - e.flow <= 0) continue;
44                 if (dist[e.u] > dist[e.v] + e.cost) {
45                     dist[e.u] = dist[e.v] + e.cost;
46                     if (!inqueue[e.u]) {
47                         Q.push(e.u);
48                         inqueue[e.u] = true;
49                     }
50                 }
51             }
52         }
53
54         return dist[t] != INF;
55     }
56
57     int cost = 0;
58     vector<int> ptr;
59     int dfs(int v, int f) {
60         if (v == t || f == 0) return f;
61         for (auto &cid = ptr[v]; cid < size(adj[v]);) {
62             auto eid = adj[v][cid];
63             auto &e = edges[eid];
64             cid++;
65             if (e.cap - e.flow <= 0) continue;
66             if (dist[e.v] + e.cost != dist[e.u]) continue;
67             int newf = dfs(e.u, min(f, e.cap - e.flow));
68             if (newf == 0) continue;
69             e.flow += newf;
70             edges[eid^1].flow -= newf;
71             cost += e.cost * newf;
72             return newf;
73         }
74         return 0;
75     }
76
77     int total_flow = 0;
78     int flow() {
79         while (spfa()) {
80             ptr.assign(n, 0);

```

```

81     while (int newf = dfs(s, INF))
82         total_flow += newf;
83     }
84     return total_flow;
85 }
86 };
87 //}}}

```

1.23 Eulerian Directed

```

1 // Description:
2 // Hierholzer's Algorithm
3 // An Eulerian path is a path that passes through
4 // every edge exactly once.
5 // An Eulerian circuit is an Eulerian path that
6 // starts and ends on the same node.
7
8 // An Eulerian path exists in an directed graph if
9 // the indegree and outdegree is equal
10 // for every node (not counting self-edges)
11 // except for possibly exactly one node that have
12 // outdegree - indegree = 1
13 // and one node that has indegree - outdegree = 1 (
14 // start and end nodes).
15 // An Eulerian circuit exists in an directed graph if
16 // the indegree and outdegree is equal for every
17 // node.
18
19 // The graph has to be conected (except for isolated
20 // nodes which are allowed because there
21 // are no edges connected to them).
22
23 // Problem:
24 // https://cses.fi/problemset/task/1693
25
26 // Complexity:
27 // O(E) where E is the number of edges
28
29 // How to use
30 // Check whether the path exists before trying to
31 // find it
32 // Find the root - any node that has at least 1
33 // outgoing edge
34 // (if the problem requires that you start from a
35 // node v, the root will be the node v)
36 // Count the degree;
37 //
38 // for (int i = 0; i < m; i++) {
39 //     int a, b; cin >> a >> b;
40 //     adj[a].pb(b);
41 //     root = a;
42 //     outdegree[a]++; indegree[b]++;
43 // }
44
45 // Notes
46 // It works when there are self loops, but not when
47 // there are multiple edges
48
49 vector<bool> visited;
50 vector<int> outdegree, indegree;
51 vector<vector<int>> adj, undir;
52
53 void dfs(int v) {
54     visited[v] = true;
55     for (auto u : undir[v]) {
56         if (!visited[u]) dfs(u);
57     }
58 }
59
60 int is_eulerian(int n, int root, int &start, int& end) {
61     start = -1, end = -1;
62     if (n == 1) return 2; // only one node

```

```

51     visited.assign(n + 1, false);
52     dfs(root);
53
54     for (int i = 1; i <= n; i++) {
55         if (!visited[i] && (i == n || i == 1 || outdegree
56             [i] + indegree[i] > 0)) return 0;
57     }
58
59     // start => node with indegree - outdegree = 1
60     // end => node with outdegree - indegree = 1
61     for (int i = 1; i <= n; i++) {
62         if (start == -1 && indegree[i] - outdegree[i] ==
63             1) start = i;
64         else if (end == -1 && outdegree[i] - indegree[i]
65             == 1) end = i;
66         else if (indegree[i] != outdegree[i]) return 0;
67     }
68
69     if (start == -1 && end == -1) {start = root; end =
70         root; return 2;} // has eulerian circuit and path
71     if (start != -1 && end != -1) {swap(start, end);
72         return 1;} // has eulerian path
73     return 0; // no eulerian path nor circuit
74 }
75
76 vector<int> path;
77
78 void dfs_path(int v) {
79     visited[v] = true;
80
81     while (outdegree[v] != 0) {
82         int u = adj[v][--outdegree[v]];
83         int next_edge = adj[v][outdegree[v]];
84         dfs_path(next_edge);
85     }
86     path.pb(v);
87 }
88
89 void find_path(int n, int start) {
90     path.clear();
91     visited.assign(n + 1, false);
92     dfs_path(start);
93     reverse(path.begin(), path.end());
94 }

```

1.24 Find Cycle

```

1 bitset<MAX> visited;
2 vector<int> path;
3 vector<int> adj[MAX];
4
5 bool dfs(int u, int p){
6
7     if (visited[u]) return false;
8
9     path.pb(u);
10    visited[u] = true;
11
12    for (auto v : adj[u]){
13        if (visited[v] and u != v and p != v){
14            path.pb(v); return true;
15        }
16
17        if (dfs(v, u)) return true;
18    }
19
20    path.pop_back();
21    return false;
22 }
23
24 bool has_cycle(int N){
25
26     visited.reset();

```



```

27     for (int u = 1; u <= N; ++u){
28         path.clear();
29         if (not visited[u] and dfs(u,-1))
30             return true;
31     }
32 }
33
34     return false;
35 }
36 }

```

1.25 Dinic

```

1 // Description:
2 // Obtains the maximum possible flow rate given a
  // network. A network is a graph with a single
  // source vertex and a single sink vertex in which
  // each edge has a capacity
3
4 // Problem:
5 // https://codeforces.com/gym/103708/problem/J
6
7 // Complexity:
8 //  $O(V^2 * E)$  where V is the number of vertex and E
  // is the number of edges
9
10 // Unit network
11 // A unit network is a network in which for any
  // vertex except source and sink either incoming or
  // outgoing edge is unique and has unit capacity (
  // matching problem).
12 // Complexity on unit networks:  $O(E * \sqrt{V})$ 
13
14 // Unity capacity networks
15 // A more generic settings when all edges have unit
  // capacities, but the number of incoming and
  // outgoing edges is unbounded
16 // Complexity on unity capacity networks:  $O(E * \sqrt{E})$ 
17
18 // How to use:
19 // Dinic dinic = Dinic(num_vertex, source, sink);
20 // dinic.add_edge(vertex1, vertex2, capacity);
21 // cout << dinic.max_flow() << '\n';
22
23 #include <bits/stdc++.h>
24
25 #define pb push_back
26 #define mp make_pair
27 #define pii pair<int, int>
28 #define ff first
29 #define ss second
30 #define ll long long
31
32 using namespace std;
33
34 const ll INF = 1e18+10;
35
36 struct Edge {
37     int from;
38     int to;
39     ll capacity;
40     ll flow;
41     Edge* residual;
42
43     Edge() {}
44
45     Edge(int from, int to, ll capacity) : from(from),
46     to(to), capacity(capacity) {
47         flow = 0;
48     }
49
50     ll get_capacity() {

```

```

50         return capacity - flow;
51     }
52
53     ll get_flow() {
54         return flow;
55     }
56
57     void augment(ll bottleneck) {
58         flow += bottleneck;
59         residual->flow -= bottleneck;
60     }
61
62     void reverse(ll bottleneck) {
63         flow -= bottleneck;
64         residual->flow += bottleneck;
65     }
66
67     bool operator<(const Edge& e) const {
68         return true;
69     }
70 };
71
72 struct Dinic {
73     int source;
74     int sink;
75     int nodes;
76     ll flow;
77     vector<vector<Edge*>> adj;
78     vector<int> level;
79     vector<int> next;
80     vector<int> reach;
81     vector<bool> visited;
82     vector<vector<int>> path;
83
84     Dinic(int source, int sink, int nodes) : source(
85     source), sink(sink), nodes(nodes) {
86         adj.resize(nodes + 1);
87     }
88
89     void add_edge(int from, int to, ll capacity) {
90         Edge* e1 = new Edge(from, to, capacity);
91         Edge* e2 = new Edge(to, from, 0);
92         // Edge* e2 = new Edge(to, from, capacity);
93         e1->residual = e2;
94         e2->residual = e1;
95         adj[from].pb(e1);
96         adj[to].pb(e2);
97     }
98
99     bool bfs() {
100         level.assign(nodes + 1, -1);
101         queue<int> q;
102         q.push(source);
103         level[source] = 0;
104
105         while (!q.empty()) {
106             int node = q.front();
107             q.pop();
108
109             for (auto e : adj[node]) {
110                 if (level[e->to] == -1 && e->
111                 get_capacity() > 0) {
112                     level[e->to] = level[e->from] +
113                     1;
114                     q.push(e->to);
115                 }
116             }
117         }
118
119         return level[sink] != -1;
120     }
121
122     ll dfs(int v, ll flow) {

```

```

120         if (v == sink)
121             return flow;
122
123         int sz = adj[v].size();
124         for (int i = next[v]; i < sz; i++) {
125             Edge* e = adj[v][i];
126             if (level[e->to] == level[e->from] + 1 &&
127                 e->get_capacity() > 0) {
128                 ll bottleneck = dfs(e->to, min(flow,
129                     e->get_capacity()));
130                 if (bottleneck > 0) {
131                     e->augment(bottleneck);
132                     return bottleneck;
133                 }
134             }
135             next[v] = i + 1;
136         }
137         return 0;
138     }
139
140     ll max_flow() {
141         flow = 0;
142         while(bfs()) {
143             next.assign(nodes + 1, 0);
144             ll sent = -1;
145             while (sent != 0) {
146                 sent = dfs(source, INF);
147                 flow += sent;
148             }
149         }
150         return flow;
151     }
152
153     void reachable(int v) {
154         visited[v] = true;
155
156         for (auto e : adj[v]) {
157             if (!visited[e->to] && e->get_capacity()
158 > 0) {
159                 reach.pb(e->to);
160                 visited[e->to] = true;
161                 reachable(e->to);
162             }
163         }
164
165         void print_min_cut() {
166             reach.clear();
167             visited.assign(nodes + 1, false);
168             reach.pb(source);
169             reachable(source);
170
171             for (auto v : reach) {
172                 for (auto e : adj[v]) {
173                     if (!visited[e->to] && e->
174 get_capacity() == 0) {
175                         cout << e->from << ' ' << e->to
176 << '\n';
177                     }
178                 }
179             }
180
181             ll build_path(int v, int id, ll flow) {
182                 visited[v] = true;
183                 if (v == sink) {
184                     return flow;
185                 }
186
187                 for (auto e : adj[v]) {
188                     if (!visited[e->to] && e->get_flow() > 0)
189
190 {
191                     visited[e->to] = true;
192                     ll bottleneck = build_path(e->to, id,
193 min(flow, e->get_flow()));
194                     if (bottleneck > 0) {
195                         path[id].pb(e->to);
196                         e->reverse(bottleneck);
197                         return bottleneck;
198                     }
199                 }
200             }
201
202             return 0;
203         }
204     }
205
206     void print_flow_path() {
207         path.clear();
208         ll sent = -1;
209         int id = -1;
210         while (sent != 0) {
211             visited.assign(nodes + 1, false);
212             path.pb(vector<int>{});
213             sent = build_path(source, ++id, INF);
214             path[id].pb(source);
215         }
216         path.pop_back();
217
218         for (int i = 0; i < id; i++) {
219             cout << path[i].size() << '\n';
220             reverse(path[i].begin(), path[i].end());
221             for (auto e : path[i]) {
222                 cout << e << ' ';
223             }
224             cout << '\n';
225         }
226     }
227
228     int main() {
229         ios::sync_with_stdio(false);
230         cin.tie(NULL);
231
232         int n, m; cin >> n >> m;
233
234         Dinic dinic = Dinic(1, n, n);
235
236         for (int i = 1; i <= m; i++) {
237             int v, u; cin >> v >> u;
238             dinic.add_edge(v, u, 1);
239         }
240
241         cout << dinic.max_flow() << '\n';
242         // dinic.print_min_cut();
243         // dinic.print_flow_path();
244
245         return 0;
246     }

```

1.26 Centroid Find

```

1 // Description:
2 // Indexed at zero
3 // Find a centroid, that is a node such that when it
4 // is appointed the root of the tree,
5 // each subtree has at most floor(n/2) nodes.
6 // Problem:
7 // https://cses.fi/problemset/task/2079/
8
9 // Complexity:
10 // O(n)
11
12 // How to use:

```

```

13 // get_subtree_size(0);
14 // cout << get_centroid(0) + 1 << endl;
15
16 int n;
17 vector<int> adj[MAX];
18 int subtree_size[MAX];
19
20 int get_subtree_size(int node, int par = -1) {
21     int &res = subtree_size[node];
22     res = 1;
23     for (int i : adj[node]) {
24         if (i == par) continue;
25         res += get_subtree_size(i, node);
26     }
27     return res;
28 }
29
30 int get_centroid(int node, int par = -1) {
31     for (int i : adj[node]) {
32         if (i == par) continue;
33
34         if (subtree_size[i] * 2 > n) { return
get_centroid(i, node); }
35     }
36     return node;
37 }
38
39 int main() {
40     cin >> n;
41     for (int i = 0; i < n - 1; i++) {
42         int u, v; cin >> u >> v;
43         u--; v--;
44         adj[u].push_back(v);
45         adj[v].push_back(u);
46     }
47
48     get_subtree_size(0);
49     cout << get_centroid(0) + 1 << endl;
50 }

```

2 Geometry

2.1 Closest Pair Points

```

1 // Description
2 // Find the squared distance between the closest two
   points among n points
3 // Also finds which pair of points is closest (could
   be more than one)
4
5 // Problem
6 // https://cses.fi/problemset/task/2194/
7
8 // Complexity
9 // O(n log n)
10
11 ll closest_pair_points(vp &vet){
12     pair<point, point> ans;
13     int n = vet.size();
14     sort(vet.begin(), vet.end());
15     set<point> s;
16
17     ll best_dist = LLONG_MAX;
18     int j=0;
19     for(int i=0;i<n;i++){
20         ll d = ceil(sqrt(best_dist));
21         while(j<n and vet[i].x-vet[j].x >= d){
22             s.erase(point(vet[j].y, vet[j].x));
23             j++;
24         }
25

```

```

26         auto it1 = s.lower_bound({vet[i].y - d, vet[i]
].x});
27         auto it2 = s.upper_bound({vet[i].y + d, vet[i]
].x});
28
29         for(auto it=it1; it!=it2; it++){
30             ll dx = vet[i].x - it->y;
31             ll dy = vet[i].y - it->x;
32
33             if(best_dist > dx*dx + dy*dy){
34                 best_dist = dx*dx + dy*dy;
35                 // closest pair points
36                 ans = mp(vet[i], point(it->y, it->x))
;
37             }
38         }
39
40         s.insert(point(vet[i].y, vet[i].x));
41     }
42
43     // best distance squared
44     return best_dist;
45 }

```

2.2 2d

```

1 #define vp vector<point>
2 #define ld long double
3 const ld EPS = 1e-6;
4 const ld PI = acos(-1);
5
6 // typedef ll cod;
7 // bool eq(cod a, cod b){ return (a==b); }
8 typedef ld cod;
9 bool eq(cod a, cod b){ return abs(a - b) <= EPS; }
10
11 struct point{
12     cod x, y;
13     int id;
14     point(cod x=0, cod y=0): x(x), y(y){}
15
16     point operator+(const point &o) const{ return {x+
o.x, y+o.y}; }
17     point operator-(const point &o) const{ return {x-
o.x, y-o.y}; }
18     point operator*(cod t) const{ return {x*t, y*t};
}
19     point operator/(cod t) const{ return {x/t, y/t};
}
20     cod operator*(const point &o) const{ return x * o
.x + y * o.y; }
21     cod operator^(const point &o) const{ return x * o
.y - y * o.x; }
22     bool operator<(const point &o) const{
23         return (eq(x, o.x) ? y < o.y : x < o.x);
24     }
25     bool operator==(const point &o) const{
26         return eq(x, o.x) and eq(y, o.y);
27     }
28     friend ostream& operator<<(ostream& os, point p) {
29         return os << "(" << p.x << "," << p.y << ")";
30     };
31
32     int ccw(point a, point b, point e){ // -1=dir; 0=
collinear; 1=esq;
33     cod tmp = (b-a) ^ (e-a); // vector from a to b
34     return (tmp > EPS) - (tmp < -EPS);
35 }
36
37 ld norm(point a){ // Modulo
38     return sqrt(a * a);
39 }
40 cod norm2(point a){

```

```

41     return a * a;
42 }
43 bool nulo(point a){
44     return (eq(a.x, 0) and eq(a.y, 0));
45 }
46 point rotccw(point p, ld a){
47     // a = PI*a/180; // graus
48     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+p.x*sin(a)));
49 }
50 point rot90cw(point a) { return point(a.y, -a.x); };
51 point rot90ccw(point a) { return point(-a.y, a.x); };
52
53 ld proj(point a, point b){ // a sobre b
54     return a*b/norm(b);
55 }
56 ld angle(point a, point b){ // em radianos
57     ld ang = a*b / norm(a) / norm(b);
58     return acos(max(min(ang, (ld)1), (ld)-1));
59 }
60 ld angle_vec(point v){
61     // return 180/PI*atan2(v.x, v.y); // graus
62     return atan2(v.x, v.y);
63 }
64 ld order_angle(point a, point b){ // from a to b ccw
65     (a in front of b)
66     ld aux = angle(a,b)*180/PI;
67     return ((a^b)<=0 ? aux:360-aux);
68 }
69 bool angle_less(point a1, point b1, point a2, point b2){ // ang(a1,b1) <= ang(a2,b2)
70     point p1((a1*b1), abs((a1^b1)));
71     point p2((a2*b2), abs((a2^b2)));
72     return (p1^p2) <= 0;
73 }
74 ld area(vp &p){ // (points sorted)
75     ld ret = 0;
76     for(int i=2;i<(int)p.size();i++)
77         ret += (p[i]-p[0])^(p[i-1]-p[0]);
78     return abs(ret/2);
79 }
80 ld areaT(point &a, point &b, point &c){
81     return abs((b-a)^(c-a))/2.0;
82 }
83
84 point center(vp &A){
85     point c = point();
86     int len = A.size();
87     for(int i=0;i<len;i++)
88         c=c+A[i];
89     return c/len;
90 }
91
92 point forca_mod(point p, ld m){
93     ld cm = norm(p);
94     if(cm<EPS) return point();
95     return point(p.x*m/cm,p.y*m/cm);
96 }
97
98 ld param(point a, point b, point v){
99     // v = t*(b-a) + a // return t;
100     // assert(line(a, b).inside_seg(v));
101     return ((v-a) * (b-a)) / ((b-a) * (b-a));
102 }
103
104 bool simetric(vp &a){ //ordered
105     int n = a.size();
106     point c = center(a);
107     if(n&1) return false;
108     for(int i=0;i<n/2;i++)
109         if(ccw(a[i], a[i+n/2], c) != 0)
110             return false;
111
112     return true;
113 }
114 point mirror(point m1, point m2, point p){
115     // mirror point p around segment m1m2
116     point seg = m2-m1;
117     ld t0 = ((p-m1)*seg) / (seg*seg);
118     point ort = m1 + seg*t0;
119     point pm = ort-(p-ort);
120     return pm;
121 }
122
123 // Line
124 // Line
125 // Line
126 // Line
127
128 struct line{
129     point p1, p2;
130     cod a, b, c; // ax+by+c = 0;
131     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
132     line(point p1=0, point p2=0): p1(p1), p2(p2){
133         a = p1.y - p2.y;
134         b = p2.x - p1.x;
135         c = p1 ^ p2;
136     }
137     line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
138     {
139         // Gera os pontos p1 p2 dados os coeficientes
140         // isso aqui eh um lixo mas quebra um galho
141         kkkkkk
142         if(b==0){
143             p1 = point(1, -c/a);
144             p2 = point(0, -c/a);
145         }else{
146             p1 = point(1, (-c-a*1)/b);
147             p2 = point(0, -c/b);
148         }
149     }
150     cod eval(point p){
151         return a*p.x+b*p.y+c;
152     }
153     bool inside(point p){
154         return eq(eval(p), 0);
155     }
156     point normal(){
157         return point(a, b);
158     }
159     bool inside_seg(point p){
160         return (
161             ((p1-p) ^ (p2-p)) == 0 and
162             ((p1-p) * (p2-p)) <= 0
163         );
164     }
165 }
166
167 // be careful with precision error
168 vp inter_line(line l1, line l2){
169     ld det = l1.a*l2.b - l1.b*l2.a;
170     if(det==0) return {};
171     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
172     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
173     return {point(x, y)};
174 }
175
176 // segments not collinear
177 vp inter_seg(line l1, line l2){
178     vp ans = inter_line(l1, l2);
179     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.inside_seg(ans[0]))
180         return {};
181 }

```

```

181         return {};
182     return ans;
183 }
184 bool seg_has_inter(line l1, line l2){
185     // if collinear
186     if (l1.inside_seg(l2.p1) || l1.inside_seg(l2.p2)
187         || l2.inside_seg(l1.p1) || l2.inside_seg(l1.p2))
188         return true;
189
190     return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.
191         p2, l2.p2) < 0 and
192         ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.
193         p2, l1.p2) < 0;
194 }
195 ld dist_seg(point p, point a, point b){ // point -
196     seg
197     if((p-a)*(b-a) < EPS) return norm(p-a);
198     if((p-b)*(a-b) < EPS) return norm(p-b);
199     return abs((p-a)^(b-a)) / norm(b-a);
200 }
201
202 line bisector(point a, point b){
203     point d = (b-a)*2;
204     return line(d.x, d.y, a*a - b*b);
205 }
206
207 line perpendicular(line l, point p){ // passes
208     through p
209     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
210 }
211
212 ///////////////
213 // Circle //
214 ///////////////
215
216 struct circle{
217     point c; cod r;
218     circle() : c(0, 0), r(0){}
219     circle(const point o) : c(o), r(0){}
220     circle(const point a, const point b){
221         c = (a+b)/2;
222         r = norm(a-c);
223     }
224     circle(const point a, const point b, const point
225     cc){
226         assert(ccw(a, b, cc) != 0);
227         c = inter_line(bisector(a, b), bisector(b, cc
228         ))[0];
229         r = norm(a-c);
230     }
231     bool inside(const point &a) const{
232         return norm(a - c) <= r + EPS;
233     }
234 }
235
236 pair<point, point> tangent_points(circle cr, point p)
237 {
238     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
239     point p1 = rotccw(cr.c-p, -theta);
240     point p2 = rotccw(cr.c-p, theta);
241     assert(d1 >= cr.r);
242     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
243     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
244     return {p1, p2};
245 }
246
247 circle incircle(point p1, point p2, point p3){
248     ld m1 = norm(p2-p3);
249     ld m2 = norm(p1-p3);
250     ld m3 = norm(p1-p2);
251     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
252     ld s = 0.5*(m1+m2+m3);
253     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
254     return circle(c, r);
255 }
256
257 circle circumcircle(point a, point b, point c) {
258     circle ans;
259     point u = point((b-a).y, -(b-a).x);
260     point v = point((c-a).y, -(c-a).x);
261     point n = (c-b)*0.5;
262     ld t = (u^n)/(v^u);
263     ans.c = ((a+c)*0.5) + (v*t);
264     ans.r = norm(ans.c-a);
265     return ans;
266 }
267
268 vp inter_circle_line(circle C, line L){
269     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.
270     p1)*(ab) / (ab*ab));
271     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
272     / (ab*ab);
273     if (h2 < -EPS) return {};
274     if (eq(h2, 0)) return {p};
275     point h = (ab/norm(ab)) * sqrt(h2);
276     return {p - h, p + h};
277 }
278
279 vp inter_circle(circle C1, circle C2){
280     if(C1.c == C2.c) { assert(C1.r != C2.r); return
281     {};}
282     point vec = C2.c - C1.c;
283     ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
284     ;
285     ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
286     C1.r*C1.r - p*p*d2;
287     if (sum*sum < d2 or dif*dif > d2) return {};
288     point mid = C1.c + vec*p, per = point(-vec.y, vec
289     .x) * sqrt(max((ld)0, h2) / d2);
290     if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
291     return {mid + per, mid - per};
292 }
293
294 // minimum circle cover O(n) amortizado
295 circle min_circle_cover(vp v){
296     random_shuffle(v.begin(), v.end());
297     circle ans;
298     int n = v.size();
299     for(int i=0;i<n;i++){
300         if(!ans.inside(v[i])){
301             ans = circle(v[i]);
302             for(int j=0;j<i;j++){
303                 if(!ans.inside(v[j])){
304                     ans = circle(v[i], v[j]);
305                     for(int k=0;k<j;k++){
306                         if(!ans.inside(v[k])){
307                             ans = circle(v[i], v[j], v[k]);
308                         }
309                     }
310                 }
311             }
312         }
313     }
314     return ans;
315 }

```

2.3 Inside Polygon

```

1 // Description
2 // Checks if a given point is inside, outside or on
3 // the boundary of a polygon
4
5 // Problem
6 // https://cses.fi/problemset/task/2192/

```

```

6
7 // Complexity
8 // O(n)
9
10 int inside(vp &p, point pp){
11     // 1 - inside / 0 - boundary / -1 - outside
12     int n = p.size();
13     for(int i=0;i<n;i++){
14         int j = (i+1)%n;
15         if(line({p[i], p[j]}).inside_seg(pp))
16             return 0; // boundary
17     }
18     int inter = 0;
19     for(int i=0;i<n;i++){
20         int j = (i+1)%n;
21         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
22 [i], p[j], pp)==1)
23             inter++; // up
24         else if(p[j].x <= pp.x and pp.x < p[i].x and
25 ccw(p[i], p[j], pp)==-1)
26             inter++; // down
27     }
28     if(inter%2==0) return -1; // outside
29     else return 1; // inside
30 }

```

2.4 Shoelace Boundary

```

1 // Description
2 // Shoelace formula finds the area of a polygon
3 // Boundary points return the number of integer
4 // points on the edges of a polygon
5 // not counting the vertexes
6
7 // Problem
8 // https://codeforces.com/gym/101873/problem/G
9
10 // Complexity
11 // O(n)
12
13 // before dividing by two
14 int shoelace(vector<point> & points) {
15     int n = points.size();
16     vector<point> v(n + 2);
17
18     for (int i = 1; i <= n; i++) {
19         v[i] = points[i - 1];
20     }
21     v[n + 1] = points[0];
22
23     int sum = 0;
24     for (int i = 1; i <= n; i++) {
25         sum += (v[i].x * v[i + 1].y - v[i + 1].x * v[
26 i].y);
27     }
28     sum = abs(sum);
29     return sum;
30 }
31
32 int boundary_points(vector<point> & points) {
33     int n = points.size();
34     vector<point> v(n + 2);
35
36     for (int i = 1; i <= n; i++) {
37         v[i] = points[i - 1];
38     }
39     v[n + 1] = points[0];
40
41     int ans = 0;
42     for (int i = 1; i <= n; i++) {

```

```

43         if (v[i].x == v[i + 1].x) ans += abs(v[i].y -
44 v[i + 1].y) - 1;
45         else if (v[i].y == v[i + 1].y) ans += abs(v[i
46 ].x - v[i + 1].x) - 1;
47         else ans += gcd(abs(v[i].x - v[i + 1].x), abs
48 (v[i].y - v[i + 1].y)) - 1;
49     }
50     return points.size() + ans;
51 }

```

3 Misc

3.1 Int128

```

1 __int128 read() {
2     __int128 x = 0, f = 1;
3     char ch = getchar();
4     while (ch < '0' || ch > '9') {
5         if (ch == '-') f = -1;
6         ch = getchar();
7     }
8     while (ch >= '0' && ch <= '9') {
9         x = x * 10 + ch - '0';
10        ch = getchar();
11    }
12    return x * f;
13 }
14 void print(__int128 x) {
15     if (x < 0) {
16         putchar('-');
17         x = -x;
18     }
19     if (x > 9) print(x / 10);
20     putchar(x % 10 + '0');
21 }

```

3.2 Split

```

1 vector<string> split(string txt, char key = ' '){
2     vector<string> ans;
3
4     string palTemp = "";
5     for(int i = 0; i < txt.size(); i++){
6
7         if(txt[i] == key){
8             if(palTemp.size() > 0){
9                 ans.push_back(palTemp);
10                palTemp = "";
11            }
12        } else{
13            palTemp += txt[i];
14        }
15    }
16
17    if(palTemp.size() > 0)
18        ans.push_back(palTemp);
19
20    return ans;
21 }
22 }

```

4 Data Structures

4.1 Psum2d

```

1 // Description:
2 // Queries the sum of a rectangle that goes from grid
3 [from_row][from_col] to grid[to_row][to_col]
4
5 // Problem:

```

```

5 // https://cses.fi/problemset/task/1652/
6
7 // Complexity:
8 // O(n) build
9 // O(1) query
10
11 for (int i = 1; i <= n; i++) {
12     for (int j = 1; j <= n; j++) {
13         psum[i][j] = grid[i][j] + psum[i - 1][j] + psum[i
14             ][j - 1] - psum[i - 1][j - 1];
15     }
16 }
17 while (q--) {
18     int from_row, to_row, from_col, to_col;
19     cin >> from_row >> from_col >> to_row >> to_col;
20     cout << psum[to_row][to_col] - psum[from_row - 1][
21         to_col] -
22     psum[to_row][from_col - 1] + psum[from_row - 1][
23         from_col - 1] << '\n';
24 }

```

4.2 Range Query Point Update

```

1 // Description:
2 // Indexed at zero
3 // Query - get sum of elements from range (l, r)
4 // inclusive
5 // Update - update element at position id to a value
6 // val
7 // Problem:
8 // https://codeforces.com/edu/course/2/lesson/4/1/
9 // practice/contest/273169/problem/B
10 // Complexity:
11 // O(log n) for both query and update
12 // How to use:
13 // Segtree seg = Segtree(n);
14 // seg.build(v);
15 // Notes
16 // Change neutral element and f function to perform a
17 // different operation
18 // If you want to change the operations to point
19 // query and range update
20 // Use the same segtree, but perform the following
21 // operations
22 // Query - seg.query(0, id);
23 // Update - seg.update(1, v); seg.update(r + 1, -v);
24
25 typedef long long ftype;
26
27 struct Segtree {
28     vector<ftype> seg;
29     int n;
30     const ftype NEUTRAL = 0;
31
32     Segtree(int n) {
33         int sz = 1;
34         while (sz < n) sz *= 2;
35         this->n = sz;
36
37         seg.assign(2*sz, NEUTRAL);
38     }
39
40     ftype f(ftype a, ftype b) {
41         return a + b;
42     }

```

```

43 ftype query(int pos, int ini, int fim, int p, int
44     q) {
45     if (ini >= p && fim <= q) {
46         return seg[pos];
47     }
48
49     if (q < ini || p > fim) {
50         return NEUTRAL;
51     }
52
53     int e = 2*pos + 1;
54     int d = 2*pos + 2;
55     int m = ini + (fim - ini) / 2;
56
57     return f(query(e, ini, m, p, q), query(d, m +
58         1, fim, p, q));
59 }
60
61 void update(int pos, int ini, int fim, int id,
62     int val) {
63     if (ini > id || fim < id) {
64         return;
65     }
66
67     if (ini == id && fim == id) {
68         seg[pos] = val;
69     }
70
71     return;
72
73     int e = 2*pos + 1;
74     int d = 2*pos + 2;
75     int m = ini + (fim - ini) / 2;
76
77     update(e, ini, m, id, val);
78     update(d, m + 1, fim, id, val);
79
80     seg[pos] = f(seg[e], seg[d]);
81 }
82
83 void build(int pos, int ini, int fim, vector<int>
84     &v) {
85     if (ini == fim) {
86         if (ini < (int)v.size()) {
87             seg[pos] = v[ini];
88         }
89         return;
90     }
91
92     int e = 2*pos + 1;
93     int d = 2*pos + 2;
94     int m = ini + (fim - ini) / 2;
95
96     build(e, ini, m, v);
97     build(d, m + 1, fim, v);
98
99     seg[pos] = f(seg[e], seg[d]);
100 }
101
102 ftype query(int p, int q) {
103     return query(0, 0, n - 1, p, q);
104 }
105
106 void update(int id, int val) {
107     update(0, 0, n - 1, id, val);
108 }
109
110 void build(vector<int> &v) {
111     build(0, 0, n - 1, v);
112 }
113
114 void debug() {
115     for (auto e : seg) {

```



```

112         cout << e << ' ';
113     }
114     cout << '\n';
115 }
116 };

```

4.3 Persistent

```

1 // Description:
2 // Persistent segtree allows for you to save the
   different versions of the segtree between each
   update
3 // Indexed at one
4 // Query - get sum of elements from range (l, r)
   inclusive
5 // Update - update element at position id to a value
   val
6
7 // Problem:
8 // https://cses.fi/problemset/task/1737/
9
10 // Complexity:
11 // O(log n) for both query and update
12
13 // How to use:
14 // vector<int> raiz(MAX); // vector to store the
   roots of each version
15 // Segtree seg = Segtree(INF);
16 // raiz[0] = seg.create(); // null node
17 // curr = 1; // keep track of the last version
18
19 // raiz[k] = seg.update(raiz[k], idx, val); //
   updating version k
20 // seg.query(raiz[k], l, r) // querying version k
21 // raiz[++curr] = raiz[k]; // create a new version
   based on version k
22
23 const int MAX = 2e5+17;
24 const int INF = 1e9+17;
25
26 typedef long long ftype;
27
28 struct Segtree {
29     vector<ftype> seg, d, e;
30     const ftype NEUTRAL = 0;
31     int n;
32
33     Segtree(int n) {
34         this->n = n;
35     }
36
37     ftype f(ftype a, ftype b) {
38         return a + b;
39     }
40
41     ftype create() {
42         seg.push_back(0);
43         e.push_back(0);
44         d.push_back(0);
45         return seg.size() - 1;
46     }
47
48     ftype query(int pos, int ini, int fim, int p, int
49 q) {
50     if (q < ini || p > fim) return NEUTRAL;
51     if (pos == 0) return 0;
52     if (p <= ini && fim <= q) return seg[pos];
53     int m = (ini + fim) >> 1;
54     return f(query(e[pos], ini, m, p, q), query(d
55 [pos], m + 1, fim, p, q));

```

```

56 int update(int pos, int ini, int fim, int id, int
57 val) {
58     int novo = create();
59
60     seg[novo] = seg[pos];
61     e[novo] = e[pos];
62     d[novo] = d[pos];
63
64     if (ini == fim) {
65         seg[novo] = val;
66         return novo;
67     }
68
69     int m = (ini + fim) >> 1;
70
71     if (id <= m) e[novo] = update(e[novo], ini, m
72 , id, val);
73     else d[novo] = update(d[novo], m + 1, fim, id
74 , val);
75
76     seg[novo] = f(seg[e[novo]], seg[d[novo]]);
77
78     return novo;
79 }
80
81 ftype query(int pos, int p, int q) {
82     return query(pos, 1, n, p, q);
83 }
84
85 int update(int pos, int id, int val) {
86     return update(pos, 1, n, id, val);
87 }
88 };

```

4.4 Minimum And Amount

```

1 // Description:
2 // Query - get minimum element in a range (l, r)
   inclusive
3 // and also the number of times it appears in that
   range
4 // Update - update element at position id to a value
   val
5
6 // Problem:
7 // https://codeforces.com/edu/course/2/lesson/4/1/
   practice/contest/273169/problem/C
8
9 // Complexity:
10 // O(log n) for both query and update
11
12 // How to use:
13 // Segtree seg = Segtree(n);
14 // seg.build(v);
15
16 #define pii pair<int, int>
17 #define mp make_pair
18 #define ff first
19 #define ss second
20
21 const int INF = 1e9+17;
22
23 typedef pii ftype;
24
25 struct Segtree {
26     vector<ftype> seg;
27     int n;
28     const ftype NEUTRAL = mp(INF, 0);
29
30     Segtree(int n) {
31         int sz = 1;
32         while (sz < n) sz *= 2;
33         this->n = sz;

```



```

34         seg.assign(2*sz, NEUTRAL);
35     }
36
37     ftype f(ftype a, ftype b) {
38         if (a.ff < b.ff) return a;
39         if (b.ff < a.ff) return b;
40
41         return mp(a.ff, a.ss + b.ss);
42     }
43
44     ftype query(int pos, int ini, int fim, int p, int q) {
45         if (ini >= p && fim <= q) {
46             return seg[pos];
47         }
48
49         if (q < ini || p > fim) {
50             return NEUTRAL;
51         }
52
53         int e = 2*pos + 1;
54         int d = 2*pos + 2;
55         int m = ini + (fim - ini) / 2;
56
57         return f(query(e, ini, m, p, q), query(d, m +
58 1, fim, p, q));
59     }
60
61     void update(int pos, int ini, int fim, int id,
62 int val) {
63         if (ini > id || fim < id) {
64             return;
65         }
66
67         if (ini == id && fim == id) {
68             seg[pos] = mp(val, 1);
69
70             return;
71         }
72
73         int e = 2*pos + 1;
74         int d = 2*pos + 2;
75         int m = ini + (fim - ini) / 2;
76
77         update(e, ini, m, id, val);
78         update(d, m + 1, fim, id, val);
79
80         seg[pos] = f(seg[e], seg[d]);
81     }
82
83     void build(int pos, int ini, int fim, vector<int>
84 &v) {
85         if (ini == fim) {
86             if (ini < (int)v.size()) {
87                 seg[pos] = mp(v[ini], 1);
88             }
89             return;
90         }
91
92         int e = 2*pos + 1;
93         int d = 2*pos + 2;
94         int m = ini + (fim - ini) / 2;
95
96         build(e, ini, m, v);
97         build(d, m + 1, fim, v);
98
99         seg[pos] = f(seg[e], seg[d]);
100     }
101
102     ftype query(int p, int q) {
103         return query(0, 0, n - 1, p, q);
104     }

```

```

103
104     void update(int id, int val) {
105         update(0, 0, n - 1, id, val);
106     }
107
108     void build(vector<int> &v) {
109         build(0, 0, n - 1, v);
110     }
111
112     void debug() {
113         for (auto e : seg) {
114             cout << e.ff << ' ' << e.ss << '\n';
115         }
116         cout << '\n';
117     }
118 };

```

4.5 Lazy Assignment To Segment

```

1 const long long INF = 1e18+10;
2
3 typedef long long ftype;
4
5 struct Segtree {
6     vector<ftype> seg;
7     vector<ftype> lazy;
8     int n;
9     const ftype NEUTRAL = 0;
10    const ftype NEUTRAL_LAZY = -1; // Change to -INF
    if there are negative numbers
11
12    Segtree(int n) {
13        int sz = 1;
14        // potencia de dois mais proxima
15        while (sz < n) sz *= 2;
16        this->n = sz;
17
18        // numero de nos da seg
19        seg.assign(2*sz, NEUTRAL);
20        lazy.assign(2*sz, NEUTRAL_LAZY);
21    }
22
23    ftype apply_lazy(ftype a, ftype b, int len) {
24        if (b == NEUTRAL_LAZY) return a;
25        if (a == NEUTRAL_LAZY) return b * len;
26        else return b * len;
27    }
28
29    void propagate(int pos, int ini, int fim) {
30        if (ini == fim) {
31            return;
32        }
33
34        int e = 2*pos + 1;
35        int d = 2*pos + 2;
36        int m = ini + (fim - ini) / 2;
37
38        lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
39        lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
40
41        seg[e] = apply_lazy(seg[e], lazy[pos], m -
42 ini + 1);
43        seg[d] = apply_lazy(seg[d], lazy[pos], fim -
44 m);
45
46        lazy[pos] = NEUTRAL_LAZY;
47    }
48
49    ftype f(ftype a, ftype b) {
50        return a + b;
51    }

```

```

51 ftype query(int pos, int ini, int fim, int p, int q) {
52     propagate(pos, ini, fim);
53
54     if (ini >= p && fim <= q) {
55         return seg[pos];
56     }
57
58     if (q < ini || p > fim) {
59         return NEUTRAL;
60     }
61
62     int e = 2*pos + 1;
63     int d = 2*pos + 2;
64     int m = ini + (fim - ini) / 2;
65
66     return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
67 }
68
69 void update(int pos, int ini, int fim, int p, int q, int val) {
70     propagate(pos, ini, fim);
71
72     if (ini > q || fim < p) {
73         return;
74     }
75
76     if (ini >= p && fim <= q) {
77         lazy[pos] = apply_lazy(lazy[pos], val, 1);
78         seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);
79     }
80     return;
81 }
82
83 int e = 2*pos + 1;
84 int d = 2*pos + 2;
85 int m = ini + (fim - ini) / 2;
86
87 update(e, ini, m, p, q, val);
88 update(d, m + 1, fim, p, q, val);
89
90 seg[pos] = f(seg[e], seg[d]);
91 }
92
93 void build(int pos, int ini, int fim, vector<int> &v) {
94     if (ini == fim) {
95         // se a posição existir no array original
96         // seg tamanho potencia de dois
97         if (ini < (int)v.size()) {
98             seg[pos] = v[ini];
99         }
100         return;
101     }
102
103     int e = 2*pos + 1;
104     int d = 2*pos + 2;
105     int m = ini + (fim - ini) / 2;
106
107     build(e, ini, m, v);
108     build(d, m + 1, fim, v);
109
110     seg[pos] = f(seg[e], seg[d]);
111 }
112
113 ftype query(int p, int q) {
114     return query(0, 0, n - 1, p, q);
115 }
116
117 void update(int p, int q, int val) {
118     update(0, 0, n - 1, p, q, val);
119 }
120
121 void build(vector<int> &v) {
122     build(0, 0, n - 1, v);
123 }
124
125 void debug() {
126     for (auto e : seg) {
127         cout << e << ' ';
128     }
129     cout << '\n';
130     for (auto e : lazy) {
131         cout << e << ' ';
132     }
133     cout << '\n';
134     cout << '\n';
135 }
136 };

```

4.6 Segtree2d

```

1 // Description:
2 // Indexed at zero
3 // Given a N x M grid, where i represents the row and
4 // j the column, perform the following operations
5 // update(i, j) - update the value of grid[i][j]
6 // query(i1, j1, i2, j2) - return the sum of values
7 // inside the rectangle
8 // defined by grid[i1][j1] and grid[i2][j2] inclusive
9
10 // Problem:
11 // https://cses.fi/problemset/task/1739/
12
13 // Complexity:
14 // Time complexity:
15 // O(log N * log M) for both query and update
16 // O(N * M) for build
17 // Memory complexity:
18 // 4 * M * N
19
20 // How to use:
21 // Segtree2D seg = Segtree2D(n, m);
22 // vector<vector<int>> v(n, vector<int>(m));
23 // seg.build(v);
24
25 struct Segtree2D {
26     const int MAXN = 1025;
27     const int NEUTRAL = 0;
28     int N, M;
29
30     vector<vector<int>> seg;
31
32     Segtree2D(int N, int M) {
33         this->N = N;
34         this->M = M;
35         seg.assign(4*MAXN, vector<int>(4*MAXN, NEUTRAL));
36     }
37
38     int f(int a, int b) {
39         return max(a, b);
40     }
41
42     void buildY(int noX, int lX, int rX, int noY, int lY, int rY, vector<vector<int>> &v) {
43         if (lY == rY) {
44             if (lX == rX) {
45                 seg[noX][noY] = v[rX][rY];
46             } else {
47                 seg[noX][noY] = f(seg[2*noX+1][noY], seg[2*noX+2][noY]);
48             }
49         }
50     }

```

```

47         }else{
48             int m = (lY+rY)/2;
49
50             buildY(noX, lX, rX, 2*noY+1, lY, m, v);
51             buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);
52
53             seg[noX][noY] = f(seg[noX][2*noY+1], seg[noX][2*noY+2]);
54         }
55     }
56
57     void buildX(int noX, int lX, int rX, vector<
58     vector<int>> &v){
59         if(lX != rX){
60             int m = (lX+rX)/2;
61
62             buildX(2*noX+1, lX, m, v);
63             buildX(2*noX+2, m+1, rX, v);
64         }
65
66         buildY(noX, lX, rX, 0, 0, M - 1, v);
67     }
68
69     void updateY(int noX, int lX, int rX, int noY,
70     int lY, int rY, int y){
71         if(lY == rY){
72             if(lX == rX){
73                 seg[noX][noY] = !seg[noX][noY];
74             }else{
75                 seg[noX][noY] = seg[2*noX+1][noY] +
76                 seg[2*noX+2][noY];
77             }
78         }else{
79             int m = (lY+rY)/2;
80
81             if(y <= m){
82                 updateY(noX, lX, rX, 2*noY+1, lY, m, y);
83             }else if(m < y){
84                 updateY(noX, lX, rX, 2*noY+2, m+1, rY, y);
85             }
86
87             seg[noX][noY] = seg[noX][2*noY+1] + seg[noX][2*noY+2];
88         }
89     }
90
91     void updateX(int noX, int lX, int rX, int x, int y){
92         int m = (lX+rX)/2;
93
94         if(lX != rX){
95             if(x <= m){
96                 updateX(2*noX+1, lX, m, x, y);
97             }else if(m < x){
98                 updateX(2*noX+2, m+1, rX, x, y);
99             }
100         }
101
102         updateY(noX, lX, rX, 0, 0, M - 1, y);
103     }
104
105     int queryY(int noX, int noY, int lY, int rY, int aY, int bY){
106         if(aY <= lY && rY <= bY) return seg[noX][noY];
107     };
108
109     int m = (lY+rY)/2;
110
111     if(bY <= m) return queryY(noX, 2*noY+1, lY, m, aY, bY);
112     if(m < aY) return queryY(noX, 2*noY+2, m+1, rY, aY, bY);
113
114     return f(queryY(noX, 2*noY+1, lY, m, aY, bY), queryY(noX, 2*noY+2, m+1, rY, aY, bY));
115 }
116
117 int queryX(int noX, int lX, int rX, int aX, int bX, int aY, int bY){
118     if(aX <= lX && rX <= bX) return queryY(noX, 0, 0, M - 1, aY, bY);
119
120     int m = (lX+rX)/2;
121
122     if(bX <= m) return queryX(2*noX+1, lX, m, aX, bX, aY, bY);
123     if(m < aX) return queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
124
125     return f(queryX(2*noX+1, lX, m, aX, bX, aY, bY), queryX(2*noX+2, m+1, rX, aX, bX, aY, bY));
126 }
127
128 void build(vector<vector<int>> &v) {
129     buildX(0, 0, N - 1, v);
130 }
131
132 int query(int aX, int aY, int bX, int bY) {
133     return queryX(0, 0, N - 1, aX, bX, aY, bY);
134 }
135
136 void update(int x, int y) {
137     updateX(0, 0, N - 1, x, y);
138 }
139
140 };

```

4.7 Dynamic Implicit Sparse

```

1 // Description:
2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
5 // stored in an array
6 // and the queries need to be answered online so we
7 // can't sort the nodes and compress them
8 // we create nodes only when they are needed so there
9 // 'll be (Q*log(MAX)) nodes
10 // where Q is the number of queries and MAX is the
11 // maximum index a node can assume
12
13 // Query - get sum of elements from range (l, r)
14 // inclusive
15 // Update - update element at position id to a value
16 // val
17
18 // Problem:
19 // https://cses.fi/problemset/task/1648
20
21 // Complexity:
22 // O(log n) for both query and update
23
24 // How to use:
25 // MAX is the maximum index a node can assume
26
27 // Segtree seg = Segtree(MAX);
28
29 typedef long long ftype;
30
31 const int MAX = 1e9+17;
32
33 struct Segtree {
34     vector<ftype> seg, d, e;
35     const ftype NEUTRAL = 0;
36     int n;

```

```

31
32 Segtree(int n) {
33     this->n = n;
34     create();
35     create();
36 }
37
38 ftype f(ftype a, ftype b) {
39     return a + b;
40 }
41
42 ftype create() {
43     seg.push_back(0);
44     e.push_back(0);
45     d.push_back(0);
46     return seg.size() - 1;
47 }
48
49 ftype query(int pos, int ini, int fim, int p, int
50 q) {
51     if (q < ini || p > fim) return NEUTRAL;
52     if (pos == 0) return 0;
53     if (p <= ini && fim <= q) return seg[pos];
54     int m = (ini + fim) >> 1;
55     return f(query(e[pos], ini, m, p, q), query(d
56 [pos], m + 1, fim, p, q));
57 }
58
59 void update(int pos, int ini, int fim, int id,
60 int val) {
61     if (ini > id || fim < id) {
62         return;
63     }
64
65     if (ini == fim) {
66         seg[pos] = val;
67
68         return;
69     }
70
71     int m = (ini + fim) >> 1;
72
73     if (id <= m) {
74         if (e[pos] == 0) e[pos] = create();
75         update(e[pos], ini, m, id, val);
76     } else {
77         if (d[pos] == 0) d[pos] = create();
78         update(d[pos], m + 1, fim, id, val);
79     }
80
81     seg[pos] = f(seg[e[pos]], seg[d[pos]]);
82 }
83
84 ftype query(int p, int q) {
85     return query(1, 1, n, p, q);
86 }
87
88 void update(int id, int val) {
89     update(1, 1, n, id, val);
90 }
91 };

```

4.8 Segment With Maximum Sum

```

1 // Description:
2 // Query - get sum of segment that is maximum among
3 // all segments
4 // E.g
5 // Array: 5 -4 4 3 -5
6 // Maximum segment sum: 8 because 5 + (-4) + 4 + 3 =
7 // 8
8 // Update - update element at position id to a value
9 // val

```

```

7
8 // Problem:
9 // https://codeforces.com/edu/course/2/lesson/4/2/
10 // practice/contest/273278/problem/A
11
12 // Complexity:
13 // O(log n) for both query and update
14
15 // How to use:
16 // Segtree seg = Segtree(n);
17 // seg.build(v);
18
19 // Notes
20 // The maximum segment sum can be a negative number
21 // In that case, taking zero elements is the best
22 // choice
23 // So we need to take the maximum between 0 and the
24 // query
25 // max(0LL, seg.query(0, n).max_seg)
26
27 using ll = long long;
28
29 typedef ll ftype_node;
30
31 struct Node {
32     ftype_node max_seg;
33     ftype_node pref;
34     ftype_node suf;
35     ftype_node sum;
36
37     Node(ftype_node max_seg, ftype_node pref,
38         ftype_node suf, ftype_node sum) : max_seg(max_seg),
39         pref(pref), suf(suf), sum(sum) {};
40 };
41
42 typedef Node ftype;
43
44 struct Segtree {
45     vector<ftype> seg;
46     int n;
47     const ftype NEUTRAL = Node(0, 0, 0, 0);
48
49     Segtree(int n) {
50         int sz = 1;
51         // potencia de dois mais proxima
52         while (sz < n) sz *= 2;
53         this->n = sz;
54
55         // numero de nos da seg
56         seg.assign(2*sz, NEUTRAL);
57     }
58
59     ftype f(ftype a, ftype b) {
60         ftype_node max_seg = max({a.max_seg, b.
61 max_seg, a.suf + b.pref});
62         ftype_node pref = max(a.pref, a.sum + b.pref)
63 ;
64         ftype_node suf = max(b.suf, b.sum + a.suf);
65         ftype_node sum = a.sum + b.sum;
66
67         return Node(max_seg, pref, suf, sum);
68     }
69
70     ftype query(int pos, int ini, int fim, int p, int
71 q) {
72     if (ini >= p && fim <= q) {
73         return seg[pos];
74     }
75
76     if (q < ini || p > fim) {
77         return NEUTRAL;
78     }
79 }

```

```

72     int e = 2*pos + 1;
73     int d = 2*pos + 2;
74     int m = ini + (fim - ini) / 2;
75
76     return f(query(e, ini, m, p, q), query(d, m +
77 1, fim, p, q));
78 }
79
80 void update(int pos, int ini, int fim, int id,
81 int val) {
82     if (ini > id || fim < id) {
83         return;
84     }
85     if (ini == id && fim == id) {
86         seg[pos] = Node(val, val, val, val);
87     }
88     return;
89 }
90
91 int e = 2*pos + 1;
92 int d = 2*pos + 2;
93 int m = ini + (fim - ini) / 2;
94
95 update(e, ini, m, id, val);
96 update(d, m + 1, fim, id, val);
97
98 seg[pos] = f(seg[e], seg[d]);
99 }
100
101 void build(int pos, int ini, int fim, vector<int>
102 &v) {
103     if (ini == fim) {
104         // se a posição existir no array original
105         // seg tamanho potencia de dois
106         if (ini < (int)v.size()) {
107             seg[pos] = Node(v[ini], v[ini], v[ini]
108 ], v[ini]);
109         }
110         return;
111     }
112     int e = 2*pos + 1;
113     int d = 2*pos + 2;
114     int m = ini + (fim - ini) / 2;
115
116     build(e, ini, m, v);
117     build(d, m + 1, fim, v);
118
119     seg[pos] = f(seg[e], seg[d]);
120 }
121
122 ftype query(int p, int q) {
123     return query(0, 0, n - 1, p, q);
124 }
125
126 void update(int id, int val) {
127     update(0, 0, n - 1, id, val);
128 }
129
130 void build(vector<int> &v) {
131     build(0, 0, n - 1, v);
132 }
133
134 void debug() {
135     for (auto e : seg) {
136         cout << e.max_seg << ' ' << e.pref << ' '
137 << e.suf << ' ' << e.sum << '\n';
138     }
139 }

```

4.9 Lazy Addition To Segment

```

1 // Description:
2 // Query - get sum of elements from range (l, r)
3 // inclusive
4 // Update - add a value val to elements from range (
5 // 1, r) inclusive
6
7 // Problem:
8 // https://codeforces.com/edu/course/2/lesson/5/1/
9 // practice/contest/279634/problem/A
10
11 // Complexity:
12 // O(log n) for both query and update
13
14 // How to use:
15 // Segtree seg = Segtree(n);
16 // seg.build(v);
17
18 // Notes
19 // Change neutral element and f function to perform a
20 // different operation
21
22 const long long INF = 1e18+10;
23
24 typedef long long ftype;
25
26 struct Segtree {
27     vector<ftype> seg;
28     vector<ftype> lazy;
29     int n;
30     const ftype NEUTRAL = 0;
31     const ftype NEUTRAL_LAZY = -1; // change to -INF
32     if there are negative numbers
33
34     Segtree(int n) {
35         int sz = 1;
36         while (sz < n) sz *= 2;
37         this->n = sz;
38
39         seg.assign(2*sz, NEUTRAL);
40         lazy.assign(2*sz, NEUTRAL_LAZY);
41     }
42
43     ftype apply_lazy(ftype a, ftype b, int len) {
44         if (b == NEUTRAL_LAZY) return a;
45         if (a == NEUTRAL_LAZY) return b * len;
46         else return a + b * len;
47     }
48
49     void propagate(int pos, int ini, int fim) {
50         if (ini == fim) {
51             return;
52         }
53
54         int e = 2*pos + 1;
55         int d = 2*pos + 2;
56         int m = ini + (fim - ini) / 2;
57
58         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
59         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
60
61         seg[e] = apply_lazy(seg[e], lazy[pos], m -
62 ini + 1);
63         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
64 m);
65
66         lazy[pos] = NEUTRAL_LAZY;
67     }
68
69     ftype f(ftype a, ftype b) {
70         return a + b;
71     }
72 }

```

```

65 ftype query(int pos, int ini, int fim, int p, int q) {
66     propagate(pos, ini, fim);
67
68     if (ini >= p && fim <= q) {
69         return seg[pos];
70     }
71
72     if (q < ini || p > fim) {
73         return NEUTRAL;
74     }
75
76     int e = 2*pos + 1;
77     int d = 2*pos + 2;
78     int m = ini + (fim - ini) / 2;
79
80     return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
81 }
82
83 void update(int pos, int ini, int fim, int p, int q, int val) {
84     propagate(pos, ini, fim);
85
86     if (ini > q || fim < p) {
87         return;
88     }
89
90     if (ini >= p && fim <= q) {
91         lazy[pos] = apply_lazy(lazy[pos], val, 1);
92         seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);
93         return;
94     }
95
96     int e = 2*pos + 1;
97     int d = 2*pos + 2;
98     int m = ini + (fim - ini) / 2;
99
100     update(e, ini, m, p, q, val);
101     update(d, m + 1, fim, p, q, val);
102
103     seg[pos] = f(seg[e], seg[d]);
104 }
105
106 void build(int pos, int ini, int fim, vector<int> &v) {
107     if (ini == fim) {
108         if (ini < (int)v.size()) {
109             seg[pos] = v[ini];
110         }
111         return;
112     }
113
114     int e = 2*pos + 1;
115     int d = 2*pos + 2;
116     int m = ini + (fim - ini) / 2;
117
118     build(e, ini, m, v);
119     build(d, m + 1, fim, v);
120
121     seg[pos] = f(seg[e], seg[d]);
122 }
123
124 ftype query(int p, int q) {
125     return query(0, 0, n - 1, p, q);
126 }
127
128 void update(int p, int q, int val) {
129     update(0, 0, n - 1, p, q, val);
130 }
131
132 }
133
134 void build(vector<int> &v) {
135     build(0, 0, n - 1, v);
136 }
137
138 void debug() {
139     for (auto e : seg) {
140         cout << e << ' ';
141     }
142     cout << '\n';
143     for (auto e : lazy) {
144         cout << e << ' ';
145     }
146     cout << '\n';
147     cout << '\n';
148 }
149
150 }

```

4.10 Lazy Dynamic Implicit Sparse

1 // Description:
2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
5 // stored in an array
6 // and the queries need to be answered online so we
7 // can't sort the nodes and compress them
8 // we create nodes only when they are needed so there
9 // 'll be (Q*log(MAX)) nodes
10 // where Q is the number of queries and MAX is the
11 // maximum index a node can assume
12 // Query - get sum of elements from range (l, r)
13 // inclusive
14 // Update - update element at position id to a value
15 // val
16
17 // Problem:
18 // https://oj.uz/problem/view/IZh012_apple
19
20 // Complexity:
21 // O(log n) for both query and update
22
23 // How to use:
24 // MAX is the maximum index a node can assume
25 // Create a default null node
26 // Create a node to be the root of the segtree
27
28 // Segtree seg = Segtree(MAX);
29
30 const int MAX = 1e9+10;
31 const long long INF = 1e18+10;
32
33 typedef long long ftype;
34
35 struct Segtree {
36 vector<ftype> seg, d, e, lazy;
37 const ftype NEUTRAL = 0;
38 const ftype NEUTRAL_LAZY = -1; // change to -INF
39 // if the elements can be negative
40 int n;
41
42 Segtree(int n) {
43 this->n = n;
44 create();
45 create();
46 }
47
48 ftype apply_lazy(ftype a, ftype b, int len) {
49 if (b == NEUTRAL_LAZY) return a;
50 else return b * len; // change to a + b * len
51 // to add to an element instead of updating it

```

45 }
46
47 void propagate(int pos, int ini, int fim) {
48     if (seg[pos] == 0) return;
49
50     if (ini == fim) {
51         return;
52     }
53
54     int m = (ini + fim) >> 1;
55
56     if (e[pos] == 0) e[pos] = create();
57     if (d[pos] == 0) d[pos] = create();
58
59     lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[
60 pos], 1);
61     lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[
62 pos], 1);
63
64     seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[
65 pos], m - ini + 1);
66     seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[
67 pos], fim - m);
68
69     lazy[pos] = NEUTRAL_LAZY;
70 }
71
72 ftype f(ftype a, ftype b) {
73     return a + b;
74 }
75
76 ftype create() {
77     seg.push_back(0);
78     e.push_back(0);
79     d.push_back(0);
80     lazy.push_back(-1);
81     return seg.size() - 1;
82 }
83
84 ftype query(int pos, int ini, int fim, int p, int
85 q) {
86     propagate(pos, ini, fim);
87     if (q < ini || p > fim) return NEUTRAL;
88     if (pos == 0) return 0;
89     if (p <= ini && fim <= q) return seg[pos];
90     int m = (ini + fim) >> 1;
91     return f(query(e[pos], ini, m, p, q), query(d
92 [pos], m + 1, fim, p, q));
93 }
94
95 void update(int pos, int ini, int fim, int p, int
96 q, int val) {
97     propagate(pos, ini, fim);
98     if (ini > q || fim < p) {
99         return;
100     }
101
102     if (ini >= p && fim <= q) {
103         lazy[pos] = apply_lazy(lazy[pos], val, 1)
104 ;
105         seg[pos] = apply_lazy(seg[pos], val, fim
106 - ini + 1);
107
108         return;
109     }
110
111     int m = (ini + fim) >> 1;
112
113     if (e[pos] == 0) e[pos] = create();
114     update(e[pos], ini, m, p, q, val);
115
116     if (d[pos] == 0) d[pos] = create();
117     update(d[pos], m + 1, fim, p, q, val);
118 }
119
120 seg[pos] = f(seg[e[pos]], seg[d[pos]]);
121 }
122
123 ftype query(int p, int q) {
124     return query(1, 1, n, p, q);
125 }
126
127 void update(int p, int q, int val) {
128     update(1, 1, n, p, q, val);
129 }
130 };

```

4.11 Sparse Table

```

1 // Description:
2 // Data structure to query for minimum and maximum
3
4 // Problem:
5 // https://cses.fi/problemset/task/1647/
6
7 // Complexity:
8 // Build O(n log n)
9 // Query O(1)
10
11 #include <bits/stdc++.h>
12
13 using namespace std;
14
15 const int MAX = 2e5+17;
16 const int INF = 1e9+17;
17
18 struct SparseTable {
19     int n;
20     vector<int> arr;
21     vector<vector<int>>> st;
22     vector<int> log_2;
23
24     SparseTable(vector<int>& arr, int& n) : arr(arr), n
25 (n) {
26         build();
27     }
28
29     void build() {
30         log_2.resize(MAX + 1);
31
32         log_2[1] = 0;
33         for (int i = 2; i <= MAX; i++) {
34             log_2[i] = log_2[i/2] + 1;
35         }
36
37         int K = log_2[n + 1];
38
39         st.resize(MAX, vector<int>(K + 1));
40
41         for (int i = 0; i < MAX; i++) {
42             for (int j = 0; j < K + 1; j++) {
43                 st[i][j] = INF;
44             }
45         }
46
47         for (int i = 0; i < n; i++) {
48             st[i][0] = arr[i];
49         }
50
51         for (int j = 1; j <= K; j++) {
52             for (int i = 0; i + (1 << j) < MAX; i++) {
53                 st[i][j] = min(st[i][j-1], st[i + (1 <<
54 j) - 1][j - 1]);
55             }
56         }
57     }
58 }

```

```

57 int query(int l, int r) {
58     int j = log2[r - l + 1];
59     return min(st[l][j], st[r - (1 << j) + 1][j]);
60 }
61 };

```

4.12 Sparse Table2d

```

1 // Description
2 // Minimum queries in a 2D grid
3
4 // Problem:
5 // https://codeforces.com/group/YgJmumGtHD/contest
  /103794/problem/D
6
7 // Complexity:
8 // Build  $O(N * M * \log(N) * \log(M))$ 
9 // Query  $O(1)$ 
10 // Memory Complexity:  $O(N * M * \log(N) * \log(M))$ 
11
12 const int MAX = 410;
13
14 struct SparseTable2D {
15     vector<vector<int>> matrix;
16     vector<vector<vector<vector<int>>>> table;
17     int n, m;
18
19     SparseTable2D(vector<vector<int>>& matrix, int n,
20         int m) : matrix(matrix), n(n), m(m) {
21         table.resize(MAX, vector<vector<vector<int>>>(MAX
22             , vector<vector<int>>(log2(MAX) + 1, vector<int>(
23                 log2(MAX) + 1)));
24         build();
25     }
26
27     int f(int a, int b) {
28         return max(a, b);
29     }
30
31     void build() {
32         for (int i = 0; i < n; i++) {
33             for (int j = 0; j < m; j++) {
34                 table[i][j][0][0] = matrix[i][j];
35             }
36         }
37
38         for (int k = 1; k <= (int)(log2(n)); k++) {
39             for (int i = 0; i + (1 << k) - 1 < n; i++) {
40                 for (int j = 0; j + (1 << k) - 1 < m; j++) {
41                     table[i][j][k][0] = f(
42                         table[i][j][k - 1][0],
43                         table[i + (1 << (k - 1))][j][k - 1][0]);
44                 }
45             }
46         }
47
48         for (int k = 1; k <= (int)(log2(m)); k++) {
49             for (int i = 0; i < n; i++) {
50                 for (int j = 0; j + (1 << k) - 1 < m; j++) {
51                     table[i][j][0][k] = f(
52                         table[i][j][0][k - 1],
53                         table[i][j + (1 << (k - 1))][0][k - 1]);
54                 }
55             }
56         }
57
58         for (int k = 1; k <= (int)(log2(n)); k++) {
59             for (int l = 1; l <= (int)(log2(m)); l++) {
60                 for (int i = 0; i + (1 << k) - 1 < n; i++) {
61                     for (int j = 0; j + (1 << l) - 1 < m; j++) {
62                         table[i][j][k][l] = f(
63                             f(

```

```

64                             table[i][j][k - 1][l - 1],
65                             table[i + (1 << (k - 1))][j][k - 1][l
66                                 - 1]
67                             ),
68                             f(
69                                 table[i][j + (1 << (l - 1))][k - 1][l
70                                     - 1],
71                                 table[i + (1 << (k - 1))][j + (1 << (
72                                     l - 1))][k - 1][l - 1])
73                             );
74                         }
75                     }
76                 }
77             }
78         }
79     }
80
81     int query(int x1, int y1, int x2, int y2) {
82         int k = log2(x2 - x1 + 1);
83         int l = log2(y2 - y1 + 1);
84
85         return f(
86             f(
87                 table[x1][y1][k][l],
88                 table[x2 - (1 << k) + 1][y1][k][l]
89             ),
90             f(
91                 table[x1][y2 - (1 << l) + 1][k][l],
92                 table[x2 - (1 << k) + 1][y2 - (1 << l) + 1][k
93                     ][l]
94             )
95         );
96     }
97 };

```

4.13 Ordered Set

```

1 // Description:
2 // insert(k) - add element k to the ordered set
3 // erase(k) - remove element k from the ordered set
4 // erase(it) - remove element it points to from the
  ordered set
5 // order_of_key(k) - returns number of elements
  strictly smaller than k
6 // find_by_order(n) - return an iterator pointing to
  the k-th element in the ordered set (counting
  from zero).
7
8 // Problem:
9 // https://cses.fi/problemset/task/2169/
10
11 // Complexity:
12 //  $O(\log n)$  for all operations
13
14 // How to use:
15 // ordered_set<int> os;
16 // cout << os.order_of_key(1) << '\n';
17 // cout << os.find_by_order(1) << '\n';
18
19 // Notes
20 // The ordered set only contains different elements
21 // By using less_equal<T> instead of less<T> on using
  ordered_set declaration
22 // The ordered_set becomes an ordered_multiset
23 // So the set can contain elements that are equal
24
25 #include <ext/pb_ds/assoc_container.hpp>
26 #include <ext/pb_ds/tree_policy.hpp>
27
28 using namespace __gnu_pbds;
29 template <typename T>
30 using ordered_set = tree<T, null_type, less<T>,
  rb_tree_tag, tree_order_statistics_node_update>;
31

```



```

32 void Erase(ordered_set<int>& a, int x){
33     int r = a.order_of_key(x);
34     auto it = a.find_by_order(r);
35     a.erase(it);
36 }

```

4.14 Priority Queue

```

1 // Description:
2 // Keeps the largest (by default) element at the top
  of the queue
3
4 // Problem:
5 // https://cses.fi/problemset/task/1164/
6
7 // Complexity:
8 // O(log n) for push and pop
9 // O(1) for looking at the element at the top
10
11 // How to use:
12 // priority_queue<int> pq;
13 // pq.push(1);
14 // pq.top();
15 // pq.pop()
16
17 // Notes
18 // To use the priority queue keeping the smallest
  element at the top
19
20 priority_queue<int, vector<int>, greater<int>> pq;

```

4.15 Two Sets

```

1 // Description
2 // The values are divided in two multisets so that
  one of them contain all values that are
3 // smaller than the median and the other one contains
  all values that are greater or equal to the
  median.
4
5 // Problem:
6 // https://atcoder.jp/contests/abc306/tasks/abc306_e
7 // Problem I - Maratona Feminina de çãProgramao da
  Unicamp 2023
8 // https://codeforces.com/group/WYIydkIPyE/contest
  /450037/attachments
9
10 // Complexity:
11 // Add and remove elements - O(log n)
12 // Return sum of biggest or smallest set or return
  the median - O(1)
13
14 using ll = long long;
15
16 struct TwoSets {
17     multiset<int> small;
18     multiset<int> big;
19     ll sums = 0;
20     ll sumb = 0;
21     int n = 0;
22
23     int size_small() {
24         return small.size();
25     }
26
27     int size_big() {
28         return big.size();
29     }
30
31     void balance() {
32         while (size_small() > n / 2) {
33             int v = *small.rbegin();

```

```

34             small.erase(prev(small.end()));
35             big.insert(v);
36             sums -= v;
37             sumb += v;
38         }
39         while (size_big() > n - n / 2) {
40             int v = *big.begin();
41             big.erase(big.begin());
42             small.insert(v);
43             sumb -= v;
44             sums += v;
45         }
46     }
47
48     void add(int x) {
49         n++;
50         small.insert(x);
51         sums += x;
52         while (!small.empty() && *small.rbegin() > *big.
  begin()) {
53             int v = *small.rbegin();
54             small.erase(prev(small.end()));
55             big.insert(v);
56             sums -= v;
57             sumb += v;
58         }
59         balance();
60     }
61
62     bool rem(int x) {
63         n--;
64         auto it1 = small.find(x);
65         auto it2 = big.find(x);
66         bool flag = false;
67         if (it1 != small.end()) {
68             sums -= *it1;
69             small.erase(it1);
70             flag = true;
71         } else if (it2 != big.end()) {
72             sumb -= *it2;
73             big.erase(it2);
74             flag = true;
75         }
76         balance();
77         return flag;
78     }
79
80     ll sum_small() {
81         return sums;
82     }
83
84     ll sum_big() {
85         return sumb;
86     }
87
88     int median() {
89         return *big.begin();
90     }
91 };

```

4.16 Dsu

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX = 1e6+17;
6
7 struct DSU {
8     int n;
9     vector<int> link, sizes;
10
11     DSU(int n) {

```

```

12     this->n = n;
13     link.assign(n+1, 0);
14     sizes.assign(n+1, 1);
15
16     for (int i = 0; i <= n; i++)
17         link[i] = i;
18 }
19
20 int find(int x) {
21     while (x != link[x])
22         x = link[x];
23
24     return x;
25 }
26
27 bool same(int a, int b) {
28     return find(a) == find(b);
29 }
30
31 void unite(int a, int b) {
32     a = find(a);
33     b = find(b);
34
35     if (a == b) return;
36
37     if (sizes[a] < sizes[b])
38         swap(a, b);
39
40     sizes[a] += sizes[b];
41     link[b] = a;
42 }
43
44 int size(int x) {
45     return sizes[x];
46 }
47 };
48
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(NULL);
52
53     int cities, roads; cin >> cities >> roads;
54     vector<int> final_roads;
55     int ans = 0;
56     DSU dsu = DSU(cities);
57     for (int i = 0, a, b; i < roads; i++) {
58         cin >> a >> b;
59         dsu.unite(a, b);
60     }
61
62     for (int i = 2; i <= cities; i++) {
63         if (!dsu.same(1, i)) {
64             ans++;
65             final_roads.push_back(i);
66             dsu.unite(1, i);
67         }
68     }
69
70     cout << ans << '\n';
71     for (auto e : final_roads) {
72         cout << "1 " << e << '\n';
73     }
74 }
75 }

```

4.17 Mergesort Tree Ordered Set

```

1 // Description:
2 // In each node, the tree keeps a sorted list of
  elements in that range.
3 // It can be used to find how many elements are
  greater than x in a given range.

```

```

4 // It can also be used to find the position of an
  element if the list was sorted.
5 // query(i, j, k) - how many elements greater than k
  are in the range (i, j)
6 // update(i, val) - changes the value of the element
  on index i to val
7
8 // Problem:
9 // https://www.beecrowd.com.br/judge/pt/problems/view
  /3097
10
11 // Complexity:
12 //  $O(n \log^2 n)$  for build
13 //  $O(\log^2 n)$  for query
14
15 #include <ext/pb_ds/assoc_container.hpp>
16 #include <ext/pb_ds/tree_policy.hpp>
17
18 using namespace __gnu_pbds;
19 template <typename T>
20 using ordered_set = tree<T, null_type, less_equal<T>,
  rb_tree_tag, tree_order_statistics_node_update>;
21
22 struct MergeSortTree {
23     vector<ordered_set<int>> tree;
24     vector<int> v;
25     int n;
26
27     MergeSortTree(int n, vector<int>& v) : n(n), v(v) {
28         int sz = 1;
29         while (sz < n) sz *= 2;
30
31         tree.resize(2 * sz);
32
33         build(0, 0, n - 1, v);
34     }
35
36     void Erase(ordered_set<int>& a, int x){
37         int r = a.order_of_key(x);
38         auto it = a.find_by_order(r);
39         a.erase(it);
40     }
41
42     ordered_set<int> merge(ordered_set<int>& a,
43         ordered_set<int>& b) {
44         ordered_set<int> res;
45
46         for (auto e : a) res.insert(e);
47         for (auto e : b) res.insert(e);
48
49         return res;
50     }
51
52     void build(int pos, int ini, int fim, vector<int>&
53         v) {
54         if (ini == fim) {
55             if (ini < (int)v.size()) {
56                 tree[pos].insert(v[ini]);
57             }
58             return;
59         }
60
61         int mid = ini + (fim - ini) / 2;
62
63         build(2 * pos + 1, ini, mid, v);
64         build(2 * pos + 2, mid + 1, fim, v);
65
66         tree[pos] = merge(tree[2 * pos + 1], tree[2 * pos
67             + 2]);
68     }
69
70     // how many elements greater than val in vector v
71     int search(ordered_set<int>& v, int val) {

```

```

69     return (int)v.size() - v.order_of_key(val + 1);
70 }
71
72 // how many elements greater than val in the range
73 // (p, q)
74 int query(int pos, int ini, int fim, int p, int q,
75           int val) {
76     if (fim < p || ini > q) {
77         return 0;
78     }
79
80     if (ini >= p && fim <= q) {
81         return search(tree[pos], val);
82     }
83
84     int mid = ini + (fim - ini) / 2;
85     return query(2 * pos + 1, ini, mid, p, q, val) +
86            query(2 * pos + 2, mid + 1, fim, p, q, val);
87 }
88
89 void update(int pos, int ini, int fim, int id, int
90            val) {
91     if (ini == id && fim == id) {
92         if (!tree[pos].empty()) Erase(tree[pos], v[id]);
93     };
94     tree[pos].insert(val);
95     return;
96 }
97
98 if (fim < id || ini > id) {
99     return;
100 }
101
102 int mid = ini + (fim - ini) / 2;
103 update(2 * pos + 1, ini, mid, id, val);
104 update(2 * pos + 2, mid + 1, fim, id, val);
105
106 if (!tree[pos].empty()) Erase(tree[pos], v[id]);
107 tree[pos].insert(val);
108 }
109
110 int query(int p, int q, int val) {
111     return query(0, 0, n - 1, p, q, val);
112 }
113
114 void update(int id, int val) {
115     update(0, 0, n - 1, id, val);
116     v[id] = val;
117 }
118
119 };

```

4.18 Mergesort Tree Vector

```

1 // Description:
2 // In each node, the tree keeps a sorted list of
3 // elements in that range.
4 // It can be used to find how many elements are
5 // greater than x in a given range.
6 // It can also be used to find the position of an
7 // element if the list was sorted.
8 // query(i, j, k) - how many elements greater than k
9 // are in the range (i, j)
10
11 // Problem:
12 // https://www.spoj.com/problems/KQUERY
13
14 // Complexity:
15 // O(n log n) for build
16 // O(log2 n) for query
17
18 struct MergeSortTree {
19     vector<vector<int>> tree;
20     int n;

```

```

MergeSortTree(int n, vector<int>& v) : n(n) {
    int sz = 1;
    while (sz < n) sz *= 2;

    tree.assign(2 * sz, vector<int>());
    build(0, 0, n - 1, v);
}

vector<int> merge(vector<int>& a, vector<int>& b) {
    vector<int> res((int)a.size() + (int)b.size());
    int it = 0, jt = 0, curr = 0;

    while (it < (int)a.size() && jt < (int)b.size())
    {
        if (a[it] <= b[jt]) {
            res[curr++] = a[it++];
        } else {
            res[curr++] = b[jt++];
        }
    }

    while (it < (int)a.size()) {
        res[curr++] = a[it++];
    }

    while (jt < (int)b.size()) {
        res[curr++] = b[jt++];
    }

    return res;
}

void build(int pos, int ini, int fim, vector<int>&
v) {
    if (ini == fim) {
        if (ini < (int)v.size()) {
            tree[pos].pb(v[ini]);
        }
        return;
    }

    int mid = ini + (fim - ini) / 2;

    build(2 * pos + 1, ini, mid, v);
    build(2 * pos + 2, mid + 1, fim, v);

    tree[pos] = merge(tree[2 * pos + 1], tree[2 * pos
+ 2]);
}

// how many elements greater than val in vector v
int search(vector<int>& v, int val) {
    auto it = upper_bound(v.begin(), v.end(), val);
    if (it == v.end()) return 0;
    return (int)v.size() - (it - v.begin());
}

// how many elements greater than val in the range
// (p, q)
int query(int pos, int ini, int fim, int p, int q,
int val) {
    if (fim < p || ini > q) {
        return 0;
    }

    if (ini >= p && fim <= q) {
        return search(tree[pos], val);
    }

    int mid = ini + (fim - ini) / 2;
    return query(2 * pos + 1, ini, mid, p, q, val) +
           query(2 * pos + 2, mid + 1, fim, p, q, val);
}

```

```

84 }
85
86 int query(int p, int q, int val) {
87     return query(0, 0, n - 1, p, q, val);
88 }
89 };

```

5 Math

5.1 Crt

```

1 ll crt(const vector<pair<ll, ll>> &vet){
2     ll ans = 0, lcm = 1;
3     ll a, b, g, x, y;
4     for(const auto &p : vet) {
5         tie(a, b) = p;
6         tie(g, x, y) = gcd(lcm, b);
7         if((a - ans) % g != 0) return -1; // no
            solution
8         ans = ans + x * ((a - ans) / g) % (b / g) *
            lcm;
9         lcm = lcm * (b / g);
10        ans = (ans % lcm + lcm) % lcm;
11    }
12    return ans;
13 }

```

5.2 Function Root

```

1 const ld EPS1 = 1e-9; // iteration precision error
2 const ld EPS2 = 1e-4; // output precision error
3
4 ld f(ld x) {
5     // exp(-x) == e^(-x)
6     return p * exp(-x) + q * sin(x) + r * cos(x) + s *
            tan(x) + t * x * x + u;
7 }
8
9 ld root(ld a, ld b) {
10    while (b - a >= EPS1) {
11        ld c = (a + b) / 2.0;
12        ld y = f(c);
13
14        if (y < 0) b = c;
15        else a = c;
16    }
17
18    return (a + b) / 2;
19 }
20
21 int main() {
22     ld ans = root(0, 1);
23     if (abs(f(ans)) <= EPS2) cout << fixed <<
            setprecision(4) << ans << '\n';
24     else cout << "No solution\n";
25
26     return 0;
27 }

```

5.3 Prime Factors

```

1 vector<pair<long long, int>> fatora(long long n) {
2     vector<pair<long long, int>> ans;
3     for(long long p = 2; p*p <= n; p++) {
4         if(n % p == 0) {
5             int expoente = 0;
6             while(n % p == 0) {
7                 n /= p;
8                 expoente++;
9             }
10            ans.emplace_back(p, expoente);

```

```

11        }
12    }
13    if(n > 1) ans.emplace_back(n, 1);
14    return ans;
15 }

```

5.4 Subsets

```

1 void subsets(vector<int>& nums){
2     int n = nums.size();
3     int powSize = 1 << n;
4
5     for(int counter = 0; counter < powSize; counter++){
6         for(int j = 0; j < n; j++) {
7             if((counter & (1LL << j)) != 0) {
8                 cout << nums[j] << ' ';
9             }
10            cout << '\n';
11        }
12    }
13 }

```

5.5 To Decimal

```

1 const string digits { "0123456789
2     ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
3
4 long long to_decimal(const string& rep, long long
5     base) {
6     long long n = 0;
7
8     for (auto c : rep) {
9         // if the number can't be represented in this
10        base
11        if (c > digits[base - 1]) return -1;
12        n *= base;
13        n += digits.find(c);
14    }
15
16    return n;
17 }

```

5.6 Multiplicative Inverse

```

1 ll extend_euclid(ll a, ll b, ll &x, ll &y) {
2     if (a == 0)
3     {
4         x = 0; y = 1;
5         return b;
6     }
7     ll x1, y1;
8     ll d = extend_euclid(b%a, a, x1, y1);
9     x = y1 - (b / a) * x1;
10    y = x1;
11    return d;
12 }
13
14 // gcd(a, m) = 1 para existir solucao
15 // ax + my = 1, ou a*x = 1 (mod m)
16 ll inv_gcd(ll a, ll m) { // com gcd
17     ll x, y;
18     extend_euclid(a, m, x, y);
19     return (((x % m) + m) % m);
20 }
21
22 ll inv(ll a, ll phim) { // com phi(m), se m for primo
23     entao phi(m) = p-1
24     ll e = phim-1;
25     return fexp(a, e, MOD);
26 }

```

5.7 Set Operations

```
1 // Complexity;
2 // O(n * m) being n and m the sizes of the two sets
3 // 2*(count1+count2)-1 (where countX is the distance
   between firstX and lastX):
4
5 vector<int> res;
6 set_union(s1.begin(), s1.end(), s2.begin(), s2.end(),
   inserter(res, res.begin()));
7 set_intersection(s1.begin(), s1.end(), s2.begin(), s2
   .end(), inserter(res, res.begin()));
8 // present in the first set, but not in the second
9 set_difference(s1.begin(), s1.end(), s2.begin(), s2.
   end(), inserter(res, res.begin()));
10 // present in one of the sets, but not in the other
11 set_symmetric_difference(s1.begin(), s1.end(), s2.
   begin(), s2.end(), inserter(res, res.begin()));
```

5.8 Representation Arbitrary Base

```
1 const string digits { "0123456789
   ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
2
3 string representation(int n, int b) {
4     string rep;
5
6     do {
7         rep.push_back(digits[n % b]);
8         n /= b;
9     } while (n);
10
11     reverse(rep.begin(), rep.end());
12
13     return rep;
14 }
```

5.9 Matrix Exponentiation

```
1 // Description:
2 // Calculate the nth term of a linear recursion
3
4 // Example Fibonacci:
5 // Given a linear recurrence, for example fibonacci
6 // F(n) = n, x <= 1
7 // F(n) = F(n - 1) + F(n - 2), x > 1
8
9 // The recurrence has two terms, so we can build a
   matrix 2 x 1 so that
10 // n + 1 = transition * n
11
12 // (2 x 1) = (2 x 2) * (2 x 1)
13 // F(n)      = a b * F(n - 1)
14 // F(n - 1)   c d   F(n - 2)
15
16 // Another Example:
17 // Given a grid 3 x n, you want to color it using 3
   distinct colors so that
18 // no adjacent place has the same color. In how many
   different ways can you do that?
19 // There are 6 ways for the first column to be
   colored using 3 distinct colors
20 // ans 6 ways using 2 equal colors and 1 distinct one
21
22 // Adding another column, there are:
23 // 3 ways to go from 2 equal to 2 equal
24 // 2 ways to go from 2 equal to 3 distinct
25 // 2 ways to go from 3 distinct to 2 equal
26 // 2 ways to go from 3 distinct to 3 distinct
27
28 // So we star with matrix 6 6 and multiply it by the
   transition 3 2 and get 18 12
```

```
29 //
   6 6
   2 2 12 12
30 // the we can exponentiate this matrix to find the
   nth column
31
32 // Problem:
33 // https://cses.fi/problemset/task/1722/
34
35 // Complexity:
36 // O(log n)
37
38 // How to use:
39 // vector<vector<ll>> v = {{1, 1}, {1, 0}};
40 // Matriz transition = Matriz(v);
41 // cout << fexp(transition, n)[0][1] << '\n';
42
43 using ll = long long;
44
45 const int MOD = 1e9+7;
46
47 struct Matriz{
48     vector<vector<ll>> mat;
49     int rows, columns;
50
51     vector<ll> operator[](int i){
52         return mat[i];
53     }
54
55     Matriz(vector<vector<ll>>& matriz){
56         mat = matriz;
57         rows = mat.size();
58         columns = mat[0].size();
59     }
60
61     Matriz(int row, int column, bool identity=false){
62         rows = row; columns = column;
63         mat.assign(rows, vector<ll>(columns, 0));
64         if(identity) {
65             for(int i = 0; i < min(rows, columns); i
66             ++){
67                 mat[i][i] = 1;
68             }
69         }
70
71     Matriz operator * (Matriz a) {
72         assert(columns == a.rows);
73         vector<vector<ll>> resp(rows, vector<ll>(a.
74         columns, 0));
75
76         for(int i = 0; i < rows; i++){
77             for(int j = 0; j < a.columns; j++){
78                 for(int k = 0; k < a.rows; k++){
79                     resp[i][j] = (resp[i][j] + (mat[i
80                     ][k] * 1LL * a[k][j]) % MOD) % MOD;
81                 }
82             }
83         }
84         return Matriz(resp);
85     }
86
87     Matriz operator + (Matriz a) {
88         assert(rows == a.rows && columns == a.columns
89         );
90         vector<vector<ll>> resp(rows, vector<ll>(
91         columns, 0));
92         for(int i = 0; i < rows; i++){
93             for(int j = 0; j < columns; j++){
94                 resp[i][j] = (resp[i][j] + mat[i][j]
95                 + a[i][j]) % MOD;
96             }
97         }
98         return Matriz(resp);
99     }
```

```

94     }
95 };
96
97 Matriz fexp(Matriz base, ll exponent){
98     Matriz result = Matriz(base.rows, base.rows, 1);
99     while(exponent > 0){
100         if(exponent & 1LL) result = result * base;
101         base = base * base;
102         exponent = exponent >> 1;
103     }
104     return result;
105 }

```

5.10 Fast Exponentiation

```

1 ll fexp(ll b, ll e, ll mod) {
2     ll res = 1;
3     b %= mod;
4     while(e){
5         if(e & 1LL)
6             res = (res * b) % mod;
7         e = e >> 1LL;
8         b = (b * b) % mod;
9     }
10    return res;
11 }

```

5.11 Phi

```

1 // Description:
2 // Euler's totient function.
3 // phi(n) is the amount of numbers in the range (1, n
4 // ) that are coprime with n
5
6 // Complexity:
7 // phi(n) - sqrt(n)
8 // phi of all numbers from 1 to n - O(n log log n)
9
10 // Properties:
11 // phi(p ^ k) = p ^ k - p ^ (k - 1)
12 // phi(p) = p - 1
13 // phi(ab) = phi(a) * phi(b) * d / phi(d) being d =
14 // gcd(a, b)
15
16 int phi(int n) {
17     int result = n;
18     for (int i = 2; i * i <= n; i++) {
19         if (n % i == 0) {
20             while (n % i == 0)
21                 n /= i;
22             result -= result / i;
23         }
24     }
25     if (n > 1)
26         result -= result / n;
27     return result;
28 }
29
30 void phi_1_to_n(int n) {
31     vector<int> phi(n + 1);
32     for (int i = 0; i <= n; i++)
33         phi[i] = i;
34
35     for (int i = 2; i <= n; i++) {
36         if (phi[i] == i) {
37             for (int j = i; j <= n; j += i)
38                 phi[j] -= phi[j] / i;
39         }
40     }
41 }

```

5.12 Binary To Decimal

```

1 int binary_to_decimal(long long n) {
2     int dec = 0, i = 0, rem;
3
4     while (n!=0) {
5         rem = n % 10;
6         n /= 10;
7         dec += rem * pow(2, i);
8         ++i;
9     }
10
11    return dec;
12 }
13
14 long long decimal_to_binary(int n) {
15     long long bin = 0;
16     int rem, i = 1;
17
18     while (n!=0) {
19         rem = n % 2;
20         n /= 2;
21         bin += rem * i;
22         i *= 10;
23     }
24
25    return bin;
26 }

```

5.13 Ceil

```

1 long long division_ceil(long long a, long long b) {
2     return 1 + ((a - 1) / b); // if a != 0
3 }

```

5.14 Horner Algorithm

```

1 // Description:
2 // Evaluates y = f(x)
3
4 // Problem:
5 // https://onlinejudge.org/index.php?option=
6 // com_onlinejudge&Itemid=8&page=show_problem&
7 // problem=439
8
9 // Complexity:
10 // O(n)
11
12 using polynomial = std::vector<int>;
13
14 polynomial p {6, -5, 2}; // p(x) = x^2 - 5x + 6;
15
16 int degree(const polynomial& p) {
17     return p.size() - 1;
18 }
19
20 int evaluate(const polynomial& p, int x) {
21     int y = 0, N = degree(p);
22
23     for (int i = N; i >= 0; --i) {
24         y *= x;
25         y += p[i];
26     }
27
28    return y;
29 }

```

5.15 Mobius

```

1 vector<int> m(MAXN, 0), lp(MAXN, 0);
2 m[1] = 1;
3 for (int i = 2; i < MAXN; ++i) {
4     if (!lp[i]) for (int j = i; j < MAXN; j += i)
5         if (!lp[j]) lp[j] = i;
6     m[i] = [&](int x) {

```

```

7         int cnt = 0;
8         while (x > 1) {
9             int k = 0, d = lp[x];
10            while (x % d == 0) {
11                x /= d;
12                ++k;
13                if (k > 1) return 0;
14            }
15            ++cnt;
16        }
17        if (cnt & 1) return -1;
18        return 1;
19    }(i);
20 }

```

5.16 Sieve Of Eratosthenes

```

1 vector<bool> is_prime(MAX, true);
2 vector<int> primes;
3
4 void sieve() {
5     is_prime[0] = is_prime[1] = false;
6     for (int i = 2; i < MAX; i++) {
7         if (is_prime[i]) {
8             primes.push_back(i);
9
10            for (int j = i + i; j < MAX; j += i)
11                is_prime[j] = false;
12        }
13    }
14 }

```

5.17 Divisors

```

1 vector<long long> all_divisors(long long n) {
2     vector<long long> ans;
3     for(long long a = 1; a*a <= n; a++){
4         if(n % a == 0) {
5             long long b = n / a;
6             ans.push_back(a);
7             if(a != b) ans.push_back(b);
8         }
9     }
10    sort(ans.begin(), ans.end());
11    return ans;
12 }

```

5.18 Linear Diophantine Equation

```

1 // int a, b, c, x1, x2, y1, y2; cin >> a >> b >> c >>
2 // x1 >> x2 >> y1 >> y2;
3 // int ans = -1;
4 // if (a == 0 && b == 0) {
5 //     if (c != 0) ans = 0;
6 //     else ans = (x2 - x1 + 1) * (y2 - y1 + 1);
7 // }
8 // else if (a == 0) {
9 //     if (c % b == 0 && y1 <= c / b && y2 >= c / b)
10 //         ans = (x2 - x1 + 1);
11 //     else ans = 0;
12 // }
13 // else if (b == 0) {
14 //     if (c % a == 0 && x1 <= c / a && x2 >= c / a)
15 //         ans = (y2 - y1 + 1);
16 //     else ans = 0;
17 // }
18 // Careful when a or b are negative or zero
19 // if (ans == -1) ans = find_all_solutions(a, b, c,
20 //     x1, x2, y1, y2);
21 // cout << ans << '\n';

```

```

20 // Problems:
21 // https://www.spoj.com/problems/CEQU/
22 // http://codeforces.com/problemsets/acmsguru/problem
23 // 99999/106
24
25 // consider trivial case a or b is 0
26 int gcd(int a, int b, int& x, int& y) {
27     if (b == 0) {
28         x = 1;
29         y = 0;
30         return a;
31     }
32     int x1, y1;
33     int d = gcd(b, a % b, x1, y1);
34     x = y1;
35     y = x1 - y1 * (a / b);
36     return d;
37 }
38
39 // x and y are one solution and g is the gcd, all
40 // passed as reference
41 // minx <= x <= maxx miny <= y <= maxy
42 bool find_any_solution(int a, int b, int c, int &x0,
43     int &y0, int &g) {
44     g = gcd(abs(a), abs(b), x0, y0);
45     if (c % g) {
46         return false;
47     }
48     x0 *= c / g;
49     y0 *= c / g;
50     if (a < 0) x0 = -x0;
51     if (b < 0) y0 = -y0;
52     return true;
53 }
54 void shift_solution(int &x, int &y, int a, int b,
55     int cnt) {
56     x += cnt * b;
57     y -= cnt * a;
58 }
59 // return number of solutions in the interval
60 int find_all_solutions(int a, int b, int c, int minx,
61     int maxx, int miny, int maxy) {
62     int x, y, g;
63     if (!find_any_solution(a, b, c, x, y, g))
64         return 0;
65     a /= g;
66     b /= g;
67
68     int sign_a = a > 0 ? +1 : -1;
69     int sign_b = b > 0 ? +1 : -1;
70
71     shift_solution(x, y, a, b, (minx - x) / b);
72     if (x < minx)
73         shift_solution(x, y, a, b, sign_b);
74     if (x > maxx)
75         return 0;
76     int lx1 = x;
77
78     shift_solution(x, y, a, b, (maxx - x) / b);
79     if (x > maxx)
80         shift_solution(x, y, a, b, -sign_b);
81     int rx1 = x;
82
83     shift_solution(x, y, a, b, -(miny - y) / a);
84     if (y < miny)
85         shift_solution(x, y, a, b, -sign_a);
86     if (y > maxy)
87         return 0;
88     int lx2 = x;

```

```

88
89     shift_solution(x, y, a, b, -(maxy - y) / a);
90     if (y > maxy)
91         shift_solution(x, y, a, b, sign_a);
92     int rx2 = x;
93
94     if (lx2 > rx2)
95         swap(lx2, rx2);
96     int lx = max(lx1, lx2);
97     int rx = min(rx1, rx2);
98
99     if (lx > rx)
100         return 0;
101     return (rx - lx) / abs(b) + 1;
102 }

```

5.19 Check If Bit Is On

```

1 // msb de 0 é undefined
2 #define msb(n) (32 - __builtin_clz(n))
3 // #define msb(n) (64 - __builtin_clzll(n))
4 // popcount
5 // turn bit off
6
7 bool bit_on(int n, int bit) {
8     if(1 & (n >> bit)) return true;
9     else return false;
10 }

```

6 Template

6.1 Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define int long long
5 #define optimize std::ios::sync_with_stdio(false);
6     cin.tie(NULL);
7 #define vi vector<int>
8 #define ll long long
9 #define pb push_back
10 #define mp make_pair
11 #define ff first
12 #define ss second
13 #define pii pair<int, int>
14 #define MOD 1000000007
15 #define sqr(x) ((x) * (x))
16 #define all(x) (x).begin(), (x).end()
17 #define FOR(i, j, n) for (int i = j; i < n; i++)
18 #define qle(i, n) (i == n ? "\n" : " ")
19 #define endl "\n"
20 const int oo = 1e9;
21 const int MAX = 1e6;
22
23 int32_t main(){ optimize;
24
25     return 0;
26 }

```

6.2 Template Clean

```

1 // Notes:
2 // Compile and execute
3 // g++ teste.cpp -o teste -std=c++17
4 // ./teste < teste.txt
5
6 // Print with precision
7 // cout << fixed << setprecision(12) << value << endl
8 ;

```

```

9 // File as input and output
10 // freopen("input.txt", "r", stdin);
11 // freopen("output.txt", "w", stdout);
12
13 #include <bits/stdc++.h>
14 using namespace std;
15
16 #define pb push_back
17 #define mp make_pair
18 #define mt make_tuple
19 #define ff first
20 #define ss second
21 #define ld long double
22 #define ll long long
23 #define int long long
24 #define pii pair<int, int>
25 #define tii tuple<int, int, int>
26
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(NULL);
30
31
32
33     return 0;
34 }

```

7 Algorithms

7.1 Delta-encoding

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int n, q;
6     cin >> n >> q;
7     int [n];
8     int delta[n+2];
9
10     while(q--){
11         int l, r, x;
12         cin >> l >> r >> x;
13         delta[l] += x;
14         delta[r+1] -= x;
15     }
16
17     int curr = 0;
18     for(int i=0; i < n; i++){
19         curr += delta[i];
20         v[i] = curr;
21     }
22
23     for(int i=0; i < n; i++){
24         cout << v[i] << ' ';
25     }
26     cout << '\n';
27
28     return 0;
29 }

```

7.2 Subsets

```

1 void subsets(vector<int>& nums){
2     int n = nums.size();
3     int powSize = 1 << n;
4
5     for(int counter = 0; counter < powSize; counter++){
6         for(int j = 0; j < n; j++){
7             if((counter & (1LL << j)) != 0) {
8                 cout << nums[j] << ' ';

```



```

9     }
10 }
11 cout << '\n';
12 }
13 }

```

7.3 Ternary Search

```

1 double ternary_search(double l, double r) {
2     double eps = 1e-9;           //set the error
3     while (r - l > eps) {
4         double m1 = l + (r - l) / 3;
5         double m2 = r - (r - l) / 3;
6         double f1 = f(m1);       //evaluates the
7         double f2 = f(m2);       //evaluates the
8         if (f1 < f2)
9             l = m1;
10        else
11            r = m2;
12    }
13    return f(l);                 //return the
14    maximum of f(x) in [l, r]

```

7.4 Biggest K

```

1 // Description: Gets sum of k biggest or k smallest
2 // elements in an array
3 // Problem: https://atcoder.jp/contests/abc306/tasks/
4 // abc306_e
5 // Complexity: O(log n)
6
7 struct SetSum {
8     ll s = 0;
9     multiset<ll> mt;
10    void add(ll x){
11        mt.insert(x);
12        s += x;
13    }
14    int pop(ll x){
15        auto f = mt.find(x);
16        if(f == mt.end()) return 0;
17        mt.erase(f);
18        s -= x;
19        return 1;
20    }
21 };
22
23 struct BigK {
24     int k;
25     SetSum gt, mt;
26     BigK(int _k){
27         k = _k;
28     }
29     void balancear(){
30         while((int)gt.mt.size() < k && (int)mt.mt.
31         size()){
32             auto p = (prev(mt.mt.end()));
33             gt.add(*p);
34             mt.pop(*p);
35         }
36         while((int)mt.mt.size() && (int)gt.mt.size()
37         &&
38         *(gt.mt.begin()) < *(prev(mt.mt.end())) ){
39             ll u = *(gt.mt.begin());
40             ll v = *(prev(mt.mt.end()));
41             gt.pop(u); mt.pop(v);

```

```

40             gt.add(v); mt.add(u);
41         }
42     }
43     void add(ll x){
44         mt.add(x);
45         balancear();
46     }
47     void rem(ll x){
48         //x = -x;
49         if(mt.pop(x) == 0)
50             gt.pop(x);
51         balancear();
52     }
53 };
54
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(NULL);
58
59     int n, k, q; cin >> n >> k >> q;
60
61     BigK big = BigK(k);
62
63     int arr[n] = {};
64
65     while (q--) {
66         int pos, num; cin >> pos >> num;
67         pos--;
68         big.rem(arr[pos]);
69         arr[pos] = num;
70         big.add(arr[pos]);
71
72         cout << big.gt.s << '\n';
73     }
74
75     return 0;
76 }

```

7.5 Binary Search First True

```

1 int first_true(int lo, int hi, function<bool(int)> f)
2 {
3     hi++;
4     while (lo < hi) {
5         int mid = lo + (hi - lo) / 2;
6         if (f(mid)) {
7             hi = mid;
8         } else {
9             lo = mid + 1;
10        }
11    }
12    return lo;

```

7.6 Binary Search Last True

```

1 int last_true(int lo, int hi, function<bool(int)> f)
2 {
3     lo--;
4     while (lo < hi) {
5         int mid = lo + (hi - lo + 1) / 2;
6         if (f(mid)) {
7             lo = mid;
8         } else {
9             hi = mid - 1;
10        }
11    }
12    return lo;

```

7.7 Lis

```

1 int lis(vector<int> const& a) {
2     int n = a.size();
3     vector<int> d(n, 1);
4     for (int i = 0; i < n; i++) {
5         for (int j = 0; j < i; j++) {
6             if (a[j] < a[i])
7                 d[i] = max(d[i], d[j] + 1);
8         }
9     }
10
11     int ans = d[0];
12     for (int i = 1; i < n; i++) {
13         ans = max(ans, d[i]);
14     }
15     return ans;
16 }

```

8 Strings

8.1 Generate All Sequences Length K

```

1 // gera todas as possíveis seqüências usando as letras
   em set (de comprimento n) e que tenham tamanho k
2 // sequence = ""
3 vector<string> generate_sequences(char set[], string
   sequence, int n, int k) {
4     if (k == 0){
5         return { sequence };
6     }
7
8     vector<string> ans;
9     for (int i = 0; i < n; i++) {
10         auto aux = generate_sequences(set, sequence +
            set[i], n, k - 1);
11         ans.insert(ans.end(), aux.begin(), aux.end());
12     };
13     // for (auto e : aux) ans.push_back(e);
14
15     return ans;
16 }

```

8.2 Lcs

```

1 // Description:
2 // Finds the longest common subsequence between two
   string
3
4 // Problem:
5 // https://codeforces.com/gym/103134/problem/B
6
7 // Complexity:
8 // O(mn) where m and n are the length of the strings
9
10 string lcsAlgo(string s1, string s2, int m, int n) {
11     int LCS_table[m + 1][n + 1];
12
13     for (int i = 0; i <= m; i++) {
14         for (int j = 0; j <= n; j++) {
15             if (i == 0 || j == 0)
16                 LCS_table[i][j] = 0;
17             else if (s1[i - 1] == s2[j - 1])
18                 LCS_table[i][j] = LCS_table[i - 1][j - 1] +
19                     1;
20             else
21                 LCS_table[i][j] = max(LCS_table[i - 1][j],
22                     LCS_table[i][j - 1]);
23         }
24     }
25
26     int index = LCS_table[m][n];

```

```

25 char lcsAlgo[index + 1];
26 lcsAlgo[index] = '\0';
27
28 int i = m, j = n;
29 while (i > 0 && j > 0) {
30     if (s1[i - 1] == s2[j - 1]) {
31         lcsAlgo[index - 1] = s1[i - 1];
32         i--;
33         j--;
34         index--;
35     }
36
37     else if (LCS_table[i - 1][j] > LCS_table[i][j -
38         1])
39         i--;
40     else
41         j--;
42 }
43 return lcsAlgo;
44 }

```

8.3 Hash

```

1 // Description:
2 // Turns a string into a integer.
3 // If the hash is different then the strings are
   different.
4 // If the hash is the same the strings may be
   different.
5
6 // Problem:
7 // https://codeforces.com/gym/104518/problem/I
8
9 // Complexity:
10 // O(n) to calculate the hash
11 // O(1) to query
12
13 // Notes:
14 // Primes 1000000007, 1000041323, 100663319,
   201326611, 1000015553, 1000028537
15
16 struct Hash {
17     const ll P = 31;
18     int n; string s;
19     vector<ll> h, hi, p;
20     Hash() {}
21     Hash(string s): s(s), n(s.size()), h(n), hi(n), p
   (n) {
22         for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1)
23             % MOD;
24         for (int i=0;i<n;i++)
25             h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
26         for (int i=n-1;i>=0;i--)
27             hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
28                 % MOD;
29     }
30     int query(int l, int r) {
31         ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
32             0));
33         return hash < 0 ? hash + MOD : hash;
34     }
35     int query_inv(int l, int r) {
36         ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
37             +1] % MOD : 0));
38         return hash < 0 ? hash + MOD : hash;
39     }
40 };

```

8.4 Trie

```

1 const int K = 26;

```

```

2
3 struct Vertex {
4     int next[K];
5     bool output = false;
6     int p = -1;
7     char pch;
8     int link = -1;
9     int go[K];
10
11     Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
12         fill(begin(next), end(next), -1);
13         fill(begin(go), end(go), -1);
14     }
15 };
16
17 vector<Vertex> t(1);
18
19 void add_string(string const& s) {
20     int v = 0;
21     for (char ch : s) {
22         int c = ch - 'a';
23         if (t[v].next[c] == -1) {
24             t[v].next[c] = t.size();
25             t.emplace_back(v, ch);
26         }
27         v = t[v].next[c];
28     }
29     t[v].output = true;
30 }
31
32 int go(int v, char ch);
33
34 int get_link(int v) {
35     if (t[v].link == -1) {
36         if (v == 0 || t[v].p == 0)
37             t[v].link = 0;
38         else
39             t[v].link = go(get_link(t[v].p), t[v].pch);
40     }
41     return t[v].link;
42 }
43
44 int go(int v, char ch) {
45     int c = ch - 'a';
46     if (t[v].go[c] == -1) {
47         if (t[v].next[c] != -1)
48             t[v].go[c] = t[v].next[c];
49         else
50             t[v].go[c] = v == 0 ? 0 : go(get_link(v),
51                                         ch);
52     }
53     return t[v].go[c];
54 }

```

8.5 Generate All Permutations

```

1 vector<string> generate_permutations(string s) {
2     int n = s.size();
3     vector<string> ans;
4
5     sort(s.begin(), s.end());
6
7     do {
8         ans.push_back(s);
9     } while (next_permutation(s.begin(), s.end()));
10
11     return ans;
12 }

```

8.6 Kmp

```

1 vector<int> prefix_function(string s) {
2     int n = (int)s.length();
3     vector<int> pi(n);
4     for (int i = 1; i < n; i++) {
5         int j = pi[i-1];
6         while (j > 0 && s[i] != s[j])
7             j = pi[j-1];
8         if (s[i] == s[j])
9             j++;
10        pi[i] = j;
11    }
12    return pi;
13 }

```

8.7 Hash2

```

1 // Hashed String {{{
2 class HashedString {
3     static const int M = (1LL << 61) - 1;
4     static const int B;
5     static vector<int> pow;
6
7     int N;
8     vector<int> p_hash;
9
10    __int128 mul(int a, int b) { return (__int128)a * b
11        ; }
12
13    int mod_mul(int a, int b) { return mul(a, b) % M; }
14
15 public:
16    explicit HashedString(string const& s) {
17        while (size(pow) < size(s) + 1) pow.push_back(
18            mod_mul(pow.back(), B));
19
20        p_hash.resize(size(s) + 1);
21        p_hash[0] = 0;
22        for (int i = 0; i < size(s); i++)
23            p_hash[i + 1] = (mul(p_hash[i], B) + s[i]) % M;
24    }
25
26    int get_hash(int l, int r) {
27        int raw_val = p_hash[r + 1] - mod_mul(p_hash[l],
28            pow[r - l + 1]);
29        return (raw_val + M) % M;
30    }
31
32    int prefix(int len) { return get_hash(0, len-1); }
33    int suffix(int len) { return get_hash(N-len, N-1); }
34
35    int whole() { return get_hash(0, N-1); }
36    int substr(int l, int len) {
37        int r = l+len-1;
38        r = min(r, N-1);
39        return get_hash(l, r);
40    }
41 };
42
43 vector<int> HashedString::pow{1};
44 mt19937 rng((uint32_t)chrono::steady_clock::now().
45     time_since_epoch().count());
46 const int HashedString::B = uniform_int_distribution<
47     int>(0, M - 1)(rng);
48 //}}}

```

8.8 Suffix Array

```

1 // Description:
2 // Suffix array is an array with the indexes of the
3 // starting letter of every
4 // suffix in an array sorted in lexicographical order
5
6 // Problem:

```

```

6 // https://codeforces.com/edu/course/2/lesson/2/1/
  practice/contest/269100/problem/A
7
8 // Complexity:
9 // O(n log n) with radix sort
10 // O(n log ^ 2 n) with regular sort
11
12 // Notes:
13 // Relevant Problems
14 // Substring search: Queries to know whether a given
  substring is present in a string
15 // Binary search for the first suffix that is greater
  or equal
16 // O(log n |p|) where |p| is the total size of the
  substrings queried
17 //
18 // Substring size: Queries to know how many times a
  given substring appears in a string
19 // Binary search both for first and last that is
  greater or equal
20 //
21 // Number of different substrings:
22 // A given suffix gives sz new substrings being sz
  the size of the suffix
23 // We can subtract the lcp (longest common prefix) to
  remove substrings
24 // that were already counted.
25 //
26 // Longest common substring between two strings:
27 // We can calculate the suffix array and lcp array of
  the two strings
28 // concatenated with a character greater than $ and
  smaller than A (like '&')
29 // The answer will be the lcp between two consecutive
  suffixes that belong to different strings
30 // (index at suffix array <= size of the first array)
31
32 void radix_sort(vector<pair<pair<int, int>, int>>& a)
33 {
34     int n = a.size();
35     vector<pair<pair<int, int>, int>> ans(n);
36
37     vector<int> count(n);
38
39     for (int i = 0; i < n; i++) {
40         count[a[i].first.second]++;
41     }
42
43     vector<int> p(n);
44
45     p[0] = 0;
46     for (int i = 1; i < n; i++) {
47         p[i] = p[i - 1] + count[i - 1];
48     }
49
50     for (int i = 0; i < n; i++) {
51         ans[p[a[i].first.second]++] = a[i];
52     }
53
54     a = ans;
55     count.assign(n, 0);
56
57     for (int i = 0; i < n; i++) {
58         count[a[i].first.first]++;
59     }
60
61     p.assign(n, 0);
62
63     p[0] = 0;
64     for (int i = 1; i < n; i++) {
65         p[i] = p[i - 1] + count[i - 1];
66     }
67
68     for (int i = 0; i < n; i++) {
69         ans[p[a[i].first.first]++] = a[i];
70     }
71
72     a = ans;
73 }
74
75 vector<int> p, c;
76
77 vector<int> suffix_array(string s) {
78     int n = s.size();
79     vector<pair<char, int>> a(n);
80     p.assign(n, 0);
81     c.assign(n, 0);
82
83     for (int i = 0; i < n; i++) {
84         a[i] = mp(s[i], i);
85     }
86
87     sort(a.begin(), a.end());
88
89     for (int i = 0; i < n; i++) {
90         p[i] = a[i].second;
91     }
92
93     c[p[0]] = 0;
94     for (int i = 1; i < n; i++) {
95         if (a[i].first == a[i - 1].first) c[p[i]] = c[p[i
  - 1]];
96         else c[p[i]] = c[p[i - 1]] + 1;
97     }
98
99     int k = 0;
100     while ((1 << k) < n) {
101         vector<pair<pair<int, int>, int>> a(n);
102         for (int i = 0; i < n; i++) {
103             a[i] = mp(mp(c[i], c[(i + (1 << k)) % n]), i);
104         }
105
106         radix_sort(a);
107
108         for (int i = 0; i < n; i++) {
109             p[i] = a[i].second;
110         }
111
112         c[p[0]] = 0;
113         for (int i = 1; i < n; i++) {
114             if (a[i].first == a[i - 1].first) c[p[i]] = c[p
  [i - 1]];
115             else c[p[i]] = c[p[i - 1]] + 1;
116         }
117
118         k++;
119     }
120
121     /* for (int i = 0; i < n; i++) {
122         for (int j = p[i]; j < n; j++) {
123             cout << s[j];
124         }
125         cout << '\n';
126     } */
127
128     return p;
129 }
130
131 // the first suffix will always be $ the (n - 1)th
  character in the string
132 vector<int> lcp_array(string s) {
133     int n = s.size();
134     vector<int> ans(n);
135     // minimum lcp
136     int k = 0;

```

```

137 for (int i = 0; i < n - 1; i++) {
138     // indice in the suffix array p of suffix
        starting in i
139     int pi = c[i];
140     // start index of the previous suffix in suffix
        array
141     int j = p[pi - 1];
142     while (s[i + k] == s[j + k]) k++;
143     ans[pi] = k;
144     k = max(k - 1, 0);
145 }
146
147 return ans;
148 }

```

8.9 Z-function

```

1 vector<int> z_function(string s) {
2     int n = (int) s.length();
3     vector<int> z(n);
4     for (int i = 1, l = 0, r = 0; i < n; ++i) {
5         if (i <= r)
6             z[i] = min(r - i + 1, z[i - l]);
7         while (i + z[i] < n && s[z[i]] == s[i + z[i]
8             ])
9             ++z[i];
10        if (i + z[i] - 1 > r)
11            l = i, r = i + z[i] - 1;
12    }
13    return z;
14 }

```

9 DP

9.1 Kadane

```

1 // Description:
2 // Finds the maximum (or minimum) sum of some
        subarray of a given array
3
4 // Problem:
5 // https://leetcode.com/problems/maximum-subarray/
        description/
6
7 // Complexity:
8 // O(n)
9
10 // Notes
11 // To solve the minimum subarray problem, start the
        variable ans with INF and change the max
        operations to min operations
12 // To not count the empty subarray as a subarray,
        start the variable ans with -INF
13 // To get the biggest possible subarray with that sum
        , change if (curr > ans) to if (curr >= ans)
14 // If the empty subarray is the answer, start and end
        will be equal to -1
15
16 int ans = 0, curr = 0;
17 int startidx = 0, start = -1, end = -1;
18
19 for (int i = 0; i < n; i++) {
20     // MAXIMUM SUBARRAY PROBLEM
21     curr = max(curr + v[i], v[i]);
22     ans = max(ans, curr);
23
24     /*
25     RECOVER INDEXES MAXIMUM SUBARRAY PROBLEM
26     if (curr + v[i] < v[i]) {
27         startidx = i;
28         curr = v[i];

```

```

29     }
30     else curr += v[i];
31
32     if (curr > ans) {
33         ans = curr;
34         start = startidx;
35         end = i;
36     }
37     /*
38
39     // MINIMUM SUBARRAY PROBLEM
40     // curr = min(curr + v[i], v[i]);
41     // ans = min(ans, curr);
42
43     /*
44     // MINIMUM SUBARRAY PROBLEM
45     if (curr + v[i] > v[i]) {
46         startidx = i;
47         curr = v[i];
48     }
49     else curr += v[i];
50
51     if (curr < ans) {
52         ans = curr;
53         start = startidx;
54         end = i;
55     }
56     /*
57 }
58
59 // cout << ans << ' ' << start << ' ' << end << '\n';

```

9.2 Edit Distance

```

1 // Description:
2 // Minimum number of operations required to transform
        a string into another
3 // Operations allowed: add character, remove
        character, replace character
4
5 // Parameters:
6 // str1 - string to be transformed into str2
7 // str2 - string that str1 will be transformed into
8 // m - size of str1
9 // n - size of str2
10
11 // Problem:
12 // https://cses.fi/problemset/task/1639
13
14 // Complexity:
15 // O(m x n)
16
17 // How to use:
18 // memset(dp, -1, sizeof(dp));
19 // string a, b;
20 // edit_distance(a, b, (int)a.size(), (int)b.size());
21
22 // Notes:
23 // Size of dp matrix is m x n
24
25 int dp[MAX][MAX];
26
27 int edit_distance(string &str1, string &str2, int m,
        int n) {
28     if (m == 0) return n;
29     if (n == 0) return m;
30
31     if (dp[m][n] != -1) return dp[m][n];
32
33     if (str1[m - 1] == str2[n - 1]) return dp[m][n] =
        edit_distance(str1, str2, m - 1, n - 1);
34     return dp[m][n] = 1 + min({edit_distance(str1,
        str2, m, n - 1), edit_distance(str1, str2, m - 1,

```

```

    n), edit_distance(str1, str2, m - 1, n - 1));
35 }

```

9.3 Coins

```

1 int tb[1005];
2 int n;
3 vector<int> moedas;
4
5 int dp(int i){
6     if(i >= n)
7         return 0;
8     if(tb[i] != -1)
9         return tb[i];
10
11     tb[i] = max(dp(i+1), dp(i+2) + moedas[i]);
12     return tb[i];
13 }
14
15 int main(){
16     memset(tb, -1, sizeof(tb));
17 }

```

9.4 Minimum Coin Change

```

1 int n;
2 vector<int> valores;
3
4 int tabela[1005];
5
6 int dp(int k){
7     if(k == 0){
8         return 0;
9     }
10     if(tabela[k] != -1)
11         return tabela[k];
12     int melhor = 1e9;
13     for(int i = 0; i < n; i++){
14         if(valores[i] <= k)
15             melhor = min(melhor, 1 + dp(k - valores[i]));
16     }
17     return tabela[k] = melhor;
18 }

```

9.5 Substr Palindrome

```

1 // êvoc deve informar se a substring de S formada
   // pelos elementos entre os índices i e j
2 // é um palindromo ou ão.
3
4 char s[MAX];
5 int calculado[MAX][MAX]; // inciado com false, ou 0
6 int tabela[MAX][MAX];
7
8 int is_palin(int i, int j){
9     if(calculado[i][j]){
10         return tabela[i][j];
11     }
12     if(i == j) return true;
13     if(i + 1 == j) return s[i] == s[j];
14
15     int ans = false;
16     if(s[i] == s[j]){
17         if(is_palin(i+1, j-1)){
18             ans = true;
19         }
20     }
21     calculado[i][j] = true;
22     tabela[i][j] = ans;
23     return ans;
24 }

```

9.6 Digits

```

1 // achar a quantidade de numeros menores que R que
   // possuem no maximo 3 digitos nao nulos
2 // a ideia eh utilizar da ordem lexicografica para
   // checar isso pois se temos por exemplo
3 // o numero 8500, a gente sabe que se pegarmos o
   // numero 7... qualquer digito depois do 7
4 // sera necessariamente menor q 8500
5
6 string r;
7 int tab[20][2][5];
8
9 // i - digito de R
10 // menor - ja pegou um numero menor que um digito de
   // R
11 // qt - quantidade de digitos nao nulos
12 int dp(int i, bool menor, int qt){
13     if(qt > 3) return 0;
14     if(i >= r.size()) return 1;
15     if(tab[i][menor][qt] != -1) return tab[i][menor][
qt];
16
17     int dr = r[i] - '0';
18     int res = 0;
19
20     for(int d = 0; d <= 9; d++) {
21         int dnn = qt + (d > 0);
22         if(menor == true) {
23             res += dp(i+1, true, dnn);
24         }
25         else if(d < dr) {
26             res += dp(i+1, true, dnn);
27         }
28         else if(d == dr) {
29             res += dp(i+1, false, dnn);
30         }
31     }
32
33     return tab[i][menor][qt] = res;
34 }

```

9.7 Knapsack With Index

```

1 void knapsack(int W, int wt[], int val[], int n) {
2     int i, w;
3     int K[n + 1][W + 1];
4
5     for (i = 0; i <= n; i++) {
6         for (w = 0; w <= W; w++) {
7             if (i == 0 || w == 0)
8                 K[i][w] = 0;
9             else if (wt[i - 1] <= w)
10                 K[i][w] = max(val[i - 1] +
K[i - 1][w - wt[i - 1]], K[i -
1][w]);
11
12             else
13                 K[i][w] = K[i - 1][w];
14         }
15     }
16
17     int res = K[n][W];
18     cout << res << endl;
19
20     w = W;
21     for (i = n; i > 0 && res > 0; i--) {
22         if (res == K[i - 1][w])
23             continue;
24         else {
25             cout << " " << wt[i - 1] ;
26             res = res - val[i - 1];
27             w = w - wt[i - 1];

```

```

28     }
29 }
30 }
31
32 int main()
33 {
34     int val[] = { 60, 100, 120 };
35     int wt[] = { 10, 20, 30 };
36     int W = 50;
37     int n = sizeof(val) / sizeof(val[0]);
38
39     knapsack(W, wt, val, n);
40
41     return 0;
42 }

```

9.8 Knapsack

```

1 int val[MAXN], peso[MAXN], dp[MAXN][MAXS];
2
3 int knapsack(int n, int m){ // n Objetos | Peso max
4     for(int i=0;i<=n;i++){
5         for(int j=0;j<=m;j++){
6             if(i==0 or j==0)
7                 dp[i][j] = 0;
8             else if(peso[i-1]<=j)
9                 dp[i][j] = max(val[i-1]+dp[i-1][j-
10                    peso[i-1]], dp[i-1][j]);
11             else
12                 dp[i][j] = dp[i-1][j];
13         }
14     }
15     return dp[n][m];
16 }

```