



Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1 DP	2		
1.1 Substr Palindrome	2		
1.2 Coins	2		
1.3 Minimum Coin Change	2		
1.4 Edit Distance	2		
1.5 Digits	2		
1.6 Knapsack With Index	3		
1.7 Kadane	3		
1.8 Knapsack	3		
2 Graphs	4		
2.1 Kruskall	4		
2.2 Prim	4		
2.3 Dijkstra	4		
2.4 Bellman Ford	5		
2.5 Bipartite	5		
2.6 Floyd Warshall	5		
2.7 Tree Diameter	6		
2.8 Centroid Decomposition	6		
2.9 Lca	6		
2.10 2sat	7		
2.11 Ford Fulkerson Edmonds Karp	8		
2.12 Dinic	9		
2.13 Find Cycle	11		
2.14 Tarjan Bridge	11		
2.15 Cycle Path Recovery	11		
2.16 Centroid Find	12		
2.17 Small To Large	12		
3 Data Structures	13		
3.1 Ordered Set	13		
3.2 Priority Queue	13		
3.3 Dsu	13		
3.4 Persistent	14		
3.5 Minimum And Amount	14		
3.6 Range Query Point Update	15		
3.7 Segment With Maximum Sum	16		
		3.8 Dynamic Implicit Sparse	17
		3.9 Lazy	18
		3.10 Lazy Dynamic Implicit Sparse	19
		3.11 Segtree2d	20
		4 Strings	21
		4.1 Lcs	21
		4.2 Kmp	21
		4.3 Trie	21
		4.4 Z-function	22
		4.5 Generate All Sequences Length K	22
		4.6 Generate All Permutations	22
		5 Algorithms	22
		5.1 Biggest K	22
		5.2 Binary Search Last True	23
		5.3 Ternary Search	23
		5.4 Delta-encoding	23
		5.5 Lis	23
		5.6 Binary Search First True	23
		6 Math	24
		6.1 Ceil	24
		6.2 Sieve Of Eratosthenes	24
		6.3 Crt	24
		6.4 Check If Bit Is On	24
		6.5 Matrix Exponentiation	24
		6.6 Fast Exponentiation	25
		6.7 Divisors	25
		6.8 Binary To Decimal	25
		6.9 Multiplicative Inverse	25
		6.10 Prime Factors	25
		6.11 Linear Diophantine Equation	25
		7 Misc	26
		7.1 Split	26
		7.2 Int128	26

1 DP

1.1 Substr Palindrome

```
1 // êvoc deve informar se a substring de S formada
  pelos elementos entre os indices i e j
2 // é um palindromo ou não.
3
4 char s[MAX];
5 int calculado[MAX][MAX]; // iniciado 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 }
```

1.2 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 }
```

1.3 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 }
```

1.4 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 matriz 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 }
```

1.5 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 }

```

1.6 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] +
11                               K[i - 1][w - wt[i - 1]], K[i -
12                               1][w]);
13             else
14                 K[i][w] = K[i - 1][w];
15         }
16     }
17
18     int res = K[n][W];
19     cout<< res << endl;
20
21     w = W;
22     for (i = n; i > 0 && res > 0; i--) {
23         if (res == K[i - 1][w])
24             continue;
25         else {
26             cout<<" "<<wt[i - 1] ;
27             res = res - val[i - 1];
28             w = w - wt[i - 1];
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     }

```

1.7 Kadane

```

1 // achar uma subsequencia continua no array que a
2 // soma seja a maior possivel
3 // nesse caso vc precisa multiplicar exatamente 1
4 // elemento da subsequencia
5 // e achar a maior soma com isso
6
7 int n, x, arr[MAX], tab[MAX][2]; // tab[maior
8 // resposta no intervalo][foi multiplicado ou ãno]
9
10 int dp(int i, bool mult) {

```

```

8     if (i == n-1) {
9         if (!mult) return arr[n-1]*x;
10        return arr[n-1];
11    }
12    if (tab[i][mult] != -1) return tab[i][mult];
13
14    int res;
15
16    if (mult) {
17        res = max(arr[i], arr[i] + dp(i+1, 1));
18    }
19    else {
20        res = max({
21            arr[i]*x,
22            arr[i]*x + dp(i+1, 1),
23            arr[i] + dp(i+1, 0)
24        });
25    }
26
27    return tab[i][mult] = res;
28 }
29
30 int main() {
31
32     memset(tab, -1, sizeof(tab));
33
34     int ans = -oo;
35     for (int i = 0; i < n; i++) {
36         ans = max(ans, dp(i, 0));
37     }
38
39     return 0;
40 }
41
42
43
44 int ans = a[0], ans_l = 0, ans_r = 0;
45 int sum = 0, minus_pos = -1;
46
47 for (int r = 0; r < n; ++r) {
48     sum += a[r];
49     if (sum > ans) {
50         ans = sum;
51         ans_l = minus_pos + 1;
52         ans_r = r;
53     }
54     if (sum < 0) {
55         sum = 0;
56         minus_pos = r;
57     }
58 }

```

1.8 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];

```

2 Graphs

2.1 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         }
70     }
71
72     return result;
73 }
```

```
68         dsu.unite(e.u, e.v);
69     }
70 }
71
72 return result;
73 }
```

2.2 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         for (int to = 0; to < n; ++to) {
35             if (adj[v][to] < min_e[to].w)
36                 min_e[to] = {adj[v][to], v};
37         }
38     }
39
40     cout << total_weight << endl;
41 }
```

2.3 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             if (d[to] > d[v] + edge.second) {
21                 d[to] = d[v] + edge.second;
22                 p[to] = v;
23                 q.insert({d[to], to});
24             }
25         }
26     }
27 }
```

```

19         int len = edge.second;
20
21         if (d[v] + len < d[to]) {
22             q.erase({d[to], to});
23             d[to] = d[v] + len;
24             p[to] = v;
25             q.insert({d[to], to});
26         }
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 }

```

2.4 Bellman Ford

```

1 struct edge
2 {
3     int a, b, cost;
4 };
5
6 int n, m, v;
7 vector<edge> e;
8 const int INF = 1000000000;
9
10 void solve()
11 {
12     vector<int> d (n, INF);
13     d[v] = 0;
14     for (int i=0; i<n-1; ++i)

```

```

15         for (int j=0; j<m; ++j)
16             if (d[e[j].a] < INF)
17                 d[e[j].b] = min (d[e[j].b], d[e[j].a]
18                     + e[j].cost);
19 }

```

2.5 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     for (int i = 1; i<=n; i++)
31         if (color[i] == NONE and not bfs(i))
32             return false;
33
34     return true;
35 }
36 }

```

2.6 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){

```

```

25         dist[i][j] = min(dist[i][j], dist
    [i][k] + dist[k][j]);
26     }
27 }
28 }
29 }
30 }

```

2.7 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 }

```

2.8 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     return u;
30 }
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 }

```

2.9 Lca

```

1 // Description:
2 // Find the lowest common ancestor between two nodes
   in a tree
3
4 // Problem:
5 // https://cses.fi/problemset/task/1688/
6
7 // Complexity:
8 //  $O(\log n)$ 
9
10 // How to use:
11 // preprocess(1);
12 // lca(a, b);
13

```

```

14 // Notes
15 // To calculate the distance between two nodes use
16 // the following formula
17 // dist[a] + dist[b] - 2*dist[lca(a, b)]
18
19 const int MAX = 2e5+17;
20
21 const int BITS = 32;
22
23 vector<int> adj[MAX];
24 // vector<pair<int, int>> adj[MAX];
25 // int dist[MAX];
26
27 int timer;
28 vector<int> tin, tout;
29 vector<vector<int>> up;
30
31 void dfs(int v, int p)
32 {
33     tin[v] = ++timer;
34     up[v][0] = p;
35
36     for (int i = 1; i <= BITS; ++i) {
37         up[v][i] = up[up[v][i-1]][i-1];
38     }
39
40     for (auto u : adj[v]) {
41         if (u != p) {
42             dfs(u, v);
43         }
44     }
45
46     /*for (auto [u, peso] : adj[v]) {
47         if (u != p) {
48             dist[u] = dist[v] + peso;
49             dfs(u, v);
50         }
51     }*/
52
53     tout[v] = ++timer;
54 }
55
56 bool is_ancestor(int u, int v)
57 {
58     return tin[u] <= tin[v] && tout[u] >= tout[v];
59 }
60
61 int lca(int u, int v)
62 {
63     if (is_ancestor(u, v))
64         return u;
65     if (is_ancestor(v, u))
66         return v;
67     for (int i = BITS; i >= 0; --i) {
68         if (!is_ancestor(up[u][i], v))
69             u = up[u][i];
70     }
71     return up[u][0];
72 }
73
74 void preprocess(int root) {
75     tin.resize(MAX);
76     tout.resize(MAX);
77     timer = 0;
78     up.assign(MAX, vector<int>(BITS + 1));
79     dfs(root, root);
80 }

```

2.10 2sat

```

1 // Description:
2 // Solves expression of the type (a v b) ^ (c v d) ^
  (e v f)

```

```

3
4 // Problem:
5 // https://cses.fi/problemset/task/1684
6
7 // Complexity:
8 // O(n + m) where n is the number of variables and m
9 // is the number of clauses
10
11 #include <bits/stdc++.h>
12 #define pb push_back
13 #define mp make_pair
14 #define pii pair<int, int>
15 #define ff first
16 #define ss second
17
18 using namespace std;
19
20 struct SAT {
21     int nodes;
22     int curr = 0;
23     int component = 0;
24     vector<vector<int>> adj;
25     vector<vector<int>> rev;
26     vector<vector<int>> condensed;
27     vector<pii> departure;
28     vector<bool> visited;
29     vector<int> scc;
30     vector<int> order;
31
32     // 1 to nodes
33     // nodes + 1 to 2 * nodes
34     SAT(int nodes) : nodes(nodes) {
35         adj.resize(2 * nodes + 1);
36         rev.resize(2 * nodes + 1);
37         visited.resize(2 * nodes + 1);
38         scc.resize(2 * nodes + 1);
39     }
40
41     void add_imp(int a, int b) {
42         adj[a].pb(b);
43         rev[b].pb(a);
44     }
45
46     int get_not(int a) {
47         if (a > nodes) return a - nodes;
48         return a + nodes;
49     }
50
51     void add_or(int a, int b) {
52         add_imp(get_not(a), b);
53         add_imp(get_not(b), a);
54     }
55
56     void add_nor(int a, int b) {
57         add_or(get_not(a), get_not(b));
58     }
59
60     void add_and(int a, int b) {
61         add_or(get_not(a), b);
62         add_or(a, get_not(b));
63         add_or(a, b);
64     }
65
66     void add_nand(int a, int b) {
67         add_or(get_not(a), b);
68         add_or(a, get_not(b));
69         add_or(get_not(a), get_not(b));
70     }
71
72     void add_xor(int a, int b) {
73         add_or(a, b);
74         add_or(get_not(a), get_not(b));
75     }
76 }

```



```

75 void add_xnor(int a, int b) {
76     add_or(get_not(a), b);
77     add_or(a, get_not(b));
78 }
79
80 void departure_time(int v) {
81     visited[v] = true;
82
83     for (auto u : adj[v]) {
84         if (!visited[u]) departure_time(u);
85     }
86
87     departure.pb(mp(++curr, v));
88 }
89
90 void find_component(int v, int component) {
91     scc[v] = component;
92     visited[v] = true;
93
94     for (auto u : rev[v]) {
95         if (!visited[u]) find_component(u,
96 component);
97     }
98 }
99
100 void topological_order(int v) {
101     visited[v] = true;
102
103     for (auto u : condensed[v]) {
104         if (!visited[u]) topological_order(u);
105     }
106
107     order.pb(v);
108 }
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 int main() {
181     ios::sync_with_stdio(false);
182     cin.tie(NULL);
183
184     int n, m; cin >> n >> m;
185
186     SAT sat = SAT(m);
187
188     for (int i = 0; i < n; i++) {
189         char op1, op2; int a, b; cin >> op1 >> a >>
190 op2 >> b;
191         if (op1 == '+' && op2 == '+') sat.add_or(a, b
192 );
193         if (op1 == '-' && op2 == '-') sat.add_or(sat.
194 get_not(a), sat.get_not(b));
195         if (op1 == '+' && op2 == '-') sat.add_or(a,
196 sat.get_not(b));
197         if (op1 == '-' && op2 == '+') sat.add_or(sat.
198 get_not(a), b);
199     }
200
201     if (!sat.is_possible()) cout << "IMPOSSIBLE\n";
202     else {
203         vector<int> ans = sat.find_ans();
204         for (int i = 1; i <= m; i++) {
205             cout << (ans[i] == 1 ? '+' : '-') << ' ';
206         }
207         cout << '\n';
208     }
209
210     return 0;
211 }

```

2.11 Ford Fulkerson Edmonds Karp

```

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 // Complexity:
5 //  $O(V * E^2)$  where V is the number of vertex and E
  is the number of edges
6
7 int n;
8 vector<vector<int>> capacity;
9 vector<vector<int>> adj;
10
11 int bfs(int s, int t, vector<int>& parent) {
12     fill(parent.begin(), parent.end(), -1);
13     parent[s] = -2;
14     queue<pair<int, int>> q;
15     q.push({s, INF});
16
17     while (!q.empty()) {
18         int cur = q.front().first;
19         int flow = q.front().second;
20         q.pop();
21
22         for (int next : adj[cur]) {
23             if (parent[next] == -1 && capacity[cur][
next]) {
24                 parent[next] = cur;
25                 int new_flow = min(flow, capacity[cur
]][next]);
26                 if (next == t)
27                     return new_flow;
28                 q.push({next, new_flow});
29             }
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 }

```

2.12 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),
to(to), capacity(capacity) {
46         flow = 0;
47     }
48
49     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) {
123         if (v == sink)
124             return flow;
125
126         int sz = adj[v].size();
127         for (int i = next[v]; i < sz; i++) {
128             Edge* e = adj[v][i];
129             if (level[e->to] == level[e->from] + 1 &&
130             e->get_capacity() > 0) {
131                 ll bottleneck = dfs(e->to, min(flow,
132                 e->get_capacity()));
133                 if (bottleneck > 0) {
134                     e->augment(bottleneck);
135                     return bottleneck;
136                 }
137             }
138             next[v] = i + 1;
139         }
140
141         return 0;
142     }
143
144     ll max_flow() {
145         flow = 0;
146         while(bfs()) {
147             next.assign(nodes + 1, 0);
148             ll sent = -1;
149
150             while (sent != 0) {
151                 sent = dfs(source, INF);
152                 flow += sent;
153             }
154             return flow;
155         }
156     }
157
158     void reachable(int v) {
159         visited[v] = true;
160
161         for (auto e : adj[v]) {
162             if (!visited[e->to] && e->get_capacity()
163             > 0) {
164                 reach.pb(e->to);
165                 visited[e->to] = true;
166                 reachable(e->to);
167             }
168         }
169     }
170
171     void print_min_cut() {
172         reach.clear();
173         visited.assign(nodes + 1, false);
174         reach.pb(source);
175         reachable(source);
176
177         for (auto v : reach) {
178             for (auto e : adj[v]) {
179                 if (!visited[e->to] && e->
180                 get_capacity() == 0) {
181                     cout << e->from << ' ' << e->to
182                     << '\n';
183                 }
184             }
185         }
186     }
187
188     ll build_path(int v, int id, ll flow) {
189         visited[v] = true;
190         if (v == sink) {
191             return flow;
192         }
193
194         for (auto e : adj[v]) {
195             if (!visited[e->to] && e->get_flow() > 0)
196             {
197                 visited[e->to] = true;
198                 ll bottleneck = build_path(e->to, id,
199                 min(flow, e->get_flow()));
200                 if (bottleneck > 0) {
201                     path[id].pb(e->to);
202                     e->reverse(bottleneck);
203                     return bottleneck;
204                 }
205             }
206         }
207
208         return 0;
209     }
210
211     void print_flow_path() {
212         path.clear();
213         ll sent = -1;
214         int id = -1;
215         while (sent != 0) {
216             visited.assign(nodes + 1, false);
217             path.pb(vector<int>{});
218             sent = build_path(source, ++id, INF);
219             path[id].pb(source);
220         }
221         path.pop_back();

```

```

213     for (int i = 0; i < id; i++) {
214         cout << path[i].size() << '\n';
215         reverse(path[i].begin(), path[i].end());
216         for (auto e : path[i]) {
217             cout << e << ' ';
218         }
219         cout << '\n';
220     }
221 }
222 };
223
224 int main() {
225     ios::sync_with_stdio(false);
226     cin.tie(NULL);
227
228     int n, m; cin >> n >> m;
229
230     Dinic dinic = Dinic(1, n, n);
231
232     for (int i = 1; i <= m; i++) {
233         int v, u; cin >> v >> u;
234         dinic.add_edge(v, u, 1);
235     }
236
237     cout << dinic.max_flow() << '\n';
238     // dinic.print_min_cut();
239     // dinic.print_flow_path();
240
241     return 0;
242 }

```

2.13 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
28     for (int u = 1; u <= N; ++u){
29         path.clear();
30         if (not visited[u] and dfs(u, -1))
31             return true;
32     }
33
34     return false;
35 }
36 }

```

2.14 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
   edge the graph is no longer connected
4
5 // Problem:
6 // https://cses.fi/problemset/task/2177/
7
8 // Complexity:
9 // O(V + E) where V is the number of vertices and E
   is the number of edges
10
11 int n;
12 vector<vector<int>> adj;
13
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 }

```

2.15 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 =
parent[v])
40             cycle.push_back(v);
41         cycle.push_back(cycle_start);
42         reverse(cycle.begin(), cycle.end());
43
44         cout << "Cycle found: ";
45         for (int v : cycle)
46             cout << v << " ";
47         cout << endl;
48     }
49 }

```

2.16 Centroid Find

```

1 // Description:
2 // Indexed at zero
3 // Find a centroid, that is a node such that when it
  is appointed the root of the tree,
4 // each subtree has at most floor(n/2) nodes.
5
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 }

```

2.17 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                 sum_num[curr] = sum_num[n];
12                 vmax[curr] = vmax[n];
13                 swap(colors[curr], colors[n]);
14             }
15
16             for (auto [item, vzs] : colors[n]) {
17                 if (colors[curr][item] + vzs > vmax[curr
]) {
18                     vmax[curr] = colors[curr][item] +
vzs;
19                     sum_num[curr] = item;
20                 }
21                 else if (colors[curr][item] + vzs ==
vmax[curr]) {
22                     sum_num[curr] += item;
23                 }
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
```

3 Data Structures

3.1 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>;
```

3.2 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;
```

3.3 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 }

3.4 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     }
57
58     int update(int pos, int ini, int fim, int id, int
59 val) {
60         int novo = create();

```

```

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

```

3.5 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
35         seg.assign(2*sz, NEUTRAL);
36     }
37

```

```

38 ftype f(ftype a, ftype b) {
39     if (a.ff < b.ff) return a;
40     if (b.ff < a.ff) return b;
41
42     return mp(a.ff, a.ss + b.ss);
43 }
44
45 ftype query(int pos, int ini, int fim, int p, int
46 q) {
47     if (ini >= p && fim <= q) {
48         return seg[pos];
49     }
50
51     if (q < ini || p > fim) {
52         return NEUTRAL;
53     }
54
55     int e = 2*pos + 1;
56     int d = 2*pos + 2;
57     int m = ini + (fim - ini) / 2;
58
59     return f(query(e, ini, m, p, q), query(d, m +
60 1, fim, p, q));
61 }
62
63 void update(int pos, int ini, int fim, int id,
64 int val) {
65     if (ini > id || fim < id) {
66         return;
67     }
68
69     if (ini == id && fim == id) {
70         seg[pos] = mp(val, 1);
71     }
72
73     return;
74
75     int e = 2*pos + 1;
76     int d = 2*pos + 2;
77     int m = ini + (fim - ini) / 2;
78
79     update(e, ini, m, id, val);
80     update(d, m + 1, fim, id, val);
81
82     seg[pos] = f(seg[e], seg[d]);
83 }
84
85 void build(int pos, int ini, int fim, vector<int>
86 &v) {
87     if (ini == fim) {
88         if (ini < (int)v.size()) {
89             seg[pos] = mp(v[ini], 1);
90         }
91         return;
92     }
93
94     int e = 2*pos + 1;
95     int d = 2*pos + 2;
96     int m = ini + (fim - ini) / 2;
97
98     build(e, ini, m, v);
99     build(d, m + 1, fim, v);
100
101     seg[pos] = f(seg[e], seg[d]);
102 }
103
104 ftype query(int p, int q) {
105     return query(0, 0, n - 1, p, q);
106 }
107
108 void update(int id, int val) {
109     update(0, 0, n - 1, id, val);
110 }

```

```

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

```

3.6 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(l, 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
44     ftype query(int pos, int ini, int fim, int p, int
45 q) {
46     if (ini >= p && fim <= q) {
47         return seg[pos];
48     }
49
50     if (q < ini || p > fim) {
51         return NEUTRAL;
52     }
53
54     int e = 2*pos + 1;
55     int d = 2*pos + 2;
56     int m = ini + (fim - ini) / 2;
57
58     query(e, ini, m, p, q);
59     query(d, m + 1, fim, p, q);
60
61     seg[pos] = f(seg[e], seg[d]);
62 }
63
64 void build(int pos, int ini, int fim, vector<int>
65 &v) {
66     if (ini == fim) {
67         if (ini < (int)v.size()) {
68             seg[pos] = mp(v[ini], 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     build(e, ini, m, v);
78     build(d, m + 1, fim, v);
79
80     seg[pos] = f(seg[e], seg[d]);
81 }
82
83 void update(int pos, int val) {
84     update(pos, val, pos, val);
85 }
86
87 void update(int id, int val) {
88     update(0, 0, n - 1, id, val);
89 }

```



```

52     int e = 2*pos + 1;
53     int d = 2*pos + 2;
54     int m = ini + (fim - ini) / 2;
55
56     return f(query(e, ini, m, p, q), query(d, m +
57 1, fim, p, q));
58 }
59
60 void update(int pos, int ini, int fim, int id,
61 int val) {
62     if (ini > id || fim < id) {
63         return;
64     }
65
66     if (ini == id && fim == id) {
67         seg[pos] = val;
68
69         return;
70     }
71
72     int e = 2*pos + 1;
73     int d = 2*pos + 2;
74     int m = ini + (fim - ini) / 2;
75
76     update(e, ini, m, id, val);
77     update(d, m + 1, fim, id, val);
78
79     seg[pos] = f(seg[e], seg[d]);
80 }
81
82 void build(int pos, int ini, int fim, vector<int>
83 &v) {
84     if (ini == fim) {
85         if (ini < (int)v.size()) {
86             seg[pos] = v[ini];
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     build(e, ini, m, v);
96     build(d, m + 1, fim, v);
97
98     seg[pos] = f(seg[e], seg[d]);
99 }
100
101 ftype query(int p, int q) {
102     return query(0, 0, n - 1, p, q);
103 }
104
105 void update(int id, int val) {
106     update(0, 0, n - 1, id, val);
107 }
108
109 void build(vector<int> &v) {
110     build(0, 0, n - 1, v);
111 }
112
113 void debug() {
114     for (auto e : seg) {
115         cout << e << ' ';
116     }
117     cout << '\n';
118 }
119 };

```

3.7 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 = 8
7 // Update - update element at position id to a value
8 // val
9
10 // Problem:
11 // https://codeforces.com/edu/course/2/lesson/4/2/
12 // practice/contest/273278/problem/A
13
14 // Complexity:
15 // O(log n) for both query and update
16
17 // How to use:
18 // Segtree seg = Segtree(n);
19 // seg.build(v);
20
21 // Notes
22 // The maximum segment sum can be a negative number
23 // In that case, taking zero elements is the best
24 // choice
25 // So we need to take the maximum between 0 and the
26 // query
27 // max(0LL, seg.query(0, n).max_seg)
28
29 using ll = long long;
30
31 typedef ll ftype_node;
32
33 struct Node {
34     ftype_node max_seg;
35     ftype_node pref;
36     ftype_node suf;
37     ftype_node sum;
38
39     Node(ftype_node max_seg, ftype_node pref,
40 ftype_node suf, ftype_node sum) : max_seg(max_seg
41 ), pref(pref), suf(suf), sum(sum) {};
42 };
43
44 typedef Node ftype;
45
46 struct Segtree {
47     vector<ftype> seg;
48     int n;
49     const ftype NEUTRAL = Node(0, 0, 0, 0);
50
51     Segtree(int n) {
52         int sz = 1;
53         // potencia de dois mais proxima
54         while (sz < n) sz *= 2;
55         this->n = sz;
56
57         // numero de nos da seg
58         seg.assign(2*sz, NEUTRAL);
59     }
60
61     ftype f(ftype a, ftype b) {
62         ftype_node max_seg = max({a.max_seg, b.
63 max_seg, a.suf + b.pref});
64         ftype_node pref = max(a.pref, a.sum + b.pref)
65 ;
66         ftype_node suf = max(b.suf, b.sum + a.suf);
67         ftype_node sum = a.sum + b.sum;
68
69         return Node(max_seg, pref, suf, sum);
70     }
71
72     ftype query(int pos, int ini, int fim, int p, int
73 q) {
74     if (ini >= p && fim <= q) {

```

```

65         return seg[pos];
66     }
67
68     if (q < ini || p > fim) {
69         return NEUTRAL;
70     }
71
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
86     if (ini == id && fim == id) {
87         seg[pos] = Node(val, val, val, val);
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
113     int e = 2*pos + 1;
114     int d = 2*pos + 2;
115     int m = ini + (fim - ini) / 2;
116
117     build(e, ini, m, v);
118     build(d, m + 1, fim, v);
119
120     seg[pos] = f(seg[e], seg[d]);
121 }
122
123 ftype query(int p, int q) {
124     return query(0, 0, n - 1, p, q);
125 }
126
127 void update(int id, int val) {
128     update(0, 0, n - 1, id, val);
129 }
130
131 void build(vector<int> &v) {
132     build(0, 0, n - 1, v);
133 }
134
135 void debug() {
136     for (auto e : seg) {

```

```

134         cout << e.max_seg << ' ' << e.pref << ' '
135 << e.suf << ' ' << e.sum << '\n';
136     }
137     cout << '\n';
138 }
139 };

```

3.8 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 // Create a default null node
27 // Create a node to be the root of the segtree
28
29 // Segtree seg = Segtree(MAX);
30 // seg.create();
31 // seg.create();
32
33 typedef long long ftype;
34
35 const int MAX = 1e9+17;
36
37 struct Segtree {
38     vector<ftype> seg, d, e, lazy;
39     const ftype NEUTRAL = 0;
40     int n;
41
42     Segtree(int n) {
43         this->n = n;
44     }
45
46     ftype f(ftype a, ftype b) {
47         return a + b;
48     }
49
50     ftype create() {
51         seg.push_back(0);
52         e.push_back(0);
53         d.push_back(0);
54         return seg.size() - 1;
55     }
56
57     ftype query(int pos, int ini, int fim, int p, int
58 q) {
59     if (q < ini || p > fim) return NEUTRAL;
60     if (pos == 0) return 0;
61     if (p <= ini && fim <= q) return seg[pos];
62     int m = (ini + fim) >> 1;
63     return f(query(e[pos], ini, m, p, q), query(d
64 [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 };

```

3.9 Lazy

```

1 // Description:
2 // Query - get sum of elements from range (l, r)
3 // inclusive
4 // Update - add a value val to elementos from range (l, r) inclusive
5 // Problem:
6 // https://codeforces.com/edu/course/2/lesson/5/1/
7 // practice/contest/279634/problem/A
8 // Complexity:
9 // O(log n) for both query and update
10
11 // How to use:
12 // Segtree seg = Segtree(n);
13 // seg.build(v);
14
15 // Notes
16 // Change neutral element and f function to perform a
17 // different operation
18
19 typedef long long ftype;
20
21 struct Segtree {
22     vector<ftype> seg;
23     vector<ftype> lazy;
24     int n;
25     const ftype NEUTRAL = 0;
26     const ftype NEUTRAL_LAZY = -1;
27
28     Segtree(int n) {
29         int sz = 1;
30         while (sz < n) sz *= 2;
31         this->n = sz;

```

```

32         seg.assign(2*sz, NEUTRAL);
33         lazy.assign(2*sz, NEUTRAL_LAZY);
34     }
35
36     ftype apply_lazy(ftype a, ftype b, int len) {
37         if (b == NEUTRAL_LAZY) return a;
38         if (a == NEUTRAL_LAZY) return b * len;
39         else return a + b * len;
40     }
41
42     void propagate(int pos, int ini, int fim) {
43         if (ini == fim) {
44             return;
45         }
46
47         int e = 2*pos + 1;
48         int d = 2*pos + 2;
49         int m = ini + (fim - ini) / 2;
50
51         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
52         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
53
54         seg[e] = apply_lazy(seg[e], lazy[pos], m -
55                             ini + 1);
56         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
57                             m);
58
59         lazy[pos] = NEUTRAL_LAZY;
60     }
61
62     ftype f(ftype a, ftype b) {
63         return a + b;
64     }
65
66     ftype query(int pos, int ini, int fim, int p, int
67               q) {
68         propagate(pos, ini, fim);
69
70         if (ini >= p && fim <= q) {
71             return seg[pos];
72         }
73
74         if (q < ini || p > fim) {
75             return NEUTRAL;
76         }
77
78         int e = 2*pos + 1;
79         int d = 2*pos + 2;
80         int m = ini + (fim - ini) / 2;
81
82         return f(query(e, ini, m, p, q), query(d, m +
83             1, fim, p, q));
84     }
85
86     void update(int pos, int ini, int fim, int p, int
87               q, int val) {
88         propagate(pos, ini, fim);
89
90         if (ini > q || fim < p) {
91             return;
92         }
93
94         if (ini >= p && fim <= q) {
95             lazy[pos] = apply_lazy(lazy[pos], val, 1);
96         }
97
98         seg[pos] = apply_lazy(seg[pos], val, fim
99             - ini + 1);
100
101         return;
102     }
103
104     int e = 2*pos + 1;
105     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 void build(vector<int> &v) {
133     build(0, 0, n - 1, v);
134 }
135
136 void debug() {
137     for (auto e : seg) {
138         cout << e << ' ';
139     }
140     cout << '\n';
141     for (auto e : lazy) {
142         cout << e << ' ';
143     }
144     cout << '\n';
145     cout << '\n';
146 }
147 };

```

3.10 Lazy Dynamic Implicit Sparse

```

1 // Description:
2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
   stored in an array
5 // and the queries need to be answered online so we
   can't sort the nodes and compress them
6 // we create nodes only when they are needed so there
   'll be (Q*log(MAX)) nodes
7 // where Q is the number of queries and MAX is the
   maximum index a node can assume
8
9 // Query - get sum of elements from range (l, r)
   inclusive
10 // Update - update element at position id to a value
   val
11
12 // Problem:
13 // https://oj.uz/problem/view/IZh012\_apple

```

```

14
15 // Complexity:
16 // O(log n) for both query and update
17
18 // How to use:
19 // MAX is the maximum index a node can assume
20 // Create a default null node
21 // Create a node to be the root of the segtree
22
23 // Segtree seg = Segtree(MAX);
24 // seg.create();
25 // seg.create();
26
27 typedef long long ftype;
28
29 const int MAX = 1e9+17;
30
31 typedef long long ftype;
32
33 const int MAX = 1e9+17;
34
35 struct Segtree {
36     vector<ftype> seg, d, e, lazy;
37     const ftype NEUTRAL = 0;
38     const ftype NEUTRAL_LAZY = -1;
39     int n;
40
41     Segtree(int n) {
42         this->n = n;
43     }
44
45     ftype apply_lazy(ftype a, ftype b, int len) {
46         if (b == NEUTRAL_LAZY) return a;
47         else return b * len;
48     }
49
50     void propagate(int pos, int ini, int fim) {
51         if (seg[pos] == 0) return;
52
53         if (ini == fim) {
54             return;
55         }
56
57         int m = (ini + fim) >> 1;
58
59         if (e[pos] == 0) e[pos] = create();
60         if (d[pos] == 0) d[pos] = create();
61
62         lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[
pos], 1);
63         lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[
pos], 1);
64
65         seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[
pos], m - ini + 1);
66         seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[
pos], fim - m);
67
68         lazy[pos] = NEUTRAL_LAZY;
69     }
70
71     ftype f(ftype a, ftype b) {
72         return a + b;
73     }
74
75     ftype create() {
76         seg.push_back(0);
77         e.push_back(0);
78         d.push_back(0);
79         lazy.push_back(-1);
80         return seg.size() - 1;
81     }
82 }

```

```

83 ftype query(int pos, int ini, int fim, int p, int q) {
84     propagate(pos, ini, fim);
85     if (q < ini || p > fim) return NEUTRAL;
86     if (pos == 0) return 0;
87     if (p <= ini && fim <= q) return seg[pos];
88     int m = (ini + fim) >> 1;
89     return f(query(e[pos], ini, m, p, q), query(d
[pos], m + 1, fim, p, q));
90 }
91
92 void update(int pos, int ini, int fim, int p, int
q, int val) {
93     propagate(pos, ini, fim);
94     if (ini > q || fim < p) {
95         return;
96     }
97
98     if (ini >= p && fim <= q) {
99         lazy[pos] = apply_lazy(lazy[pos], val, 1)
;
100         seg[pos] = apply_lazy(seg[pos], val, fim
- ini + 1);
101
102         return;
103     }
104
105     int m = (ini + fim) >> 1;
106
107     if (e[pos] == 0) e[pos] = create();
108     update(e[pos], ini, m, p, q, val);
109
110     if (d[pos] == 0) d[pos] = create();
111     update(d[pos], m + 1, fim, p, q, val);
112
113     seg[pos] = f(seg[e[pos]], seg[d[pos]]);
114 }
115
116 ftype query(int p, int q) {
117     return query(1, 1, n, p, q);
118 }
119
120 void update(int p, int q, int val) {
121     update(1, 1, n, p, q, val);
122 }
123 };

```

3.11 Segtree2d

```

1 // Description:
2 // Indexed at zero
3 // Given a N x M grid, where i represents the row and
j the column, perform the following operations
4 // update(j, i) - update the value of grid[i][j]
5 // query(j1, j2, i1, i2) - return the sum of values
inside the rectangle
6 // defined by grid[i1][j1] and grid[i2][j2] inclusive
7
8 // Problem:
9 // https://cses.fi/problemset/task/1739/
10
11 // Complexity:
12 // Time complexity:
13 // O(log N * log M) for both query and update
14 // O(N * M) for build
15 // Memory complexity:
16 // 4 * M * N
17
18 // How to use:
19 // Segtree2D seg = Segtree2D(n, n);
20 // vector<vector<int>> v(n, vector<int>(n));
21 // seg.build(v);
22
23 // Notes
24 // Indexed at zero
25
26 struct Segtree2D {
27     const int MAXN = 1025;
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.resize(2*MAXN, vector<int>(2*MAXN));
36     }
37
38     void buildY(int noX, int lX, int rX, int noY, int
lY, int rY, vector<vector<int>> &v){
39         if(lY == rY){
40             if(lX == rX){
41                 seg[noX][noY] = v[rX][rY];
42             }else{
43                 seg[noX][noY] = seg[2*noX+1][noY] +
seg[2*noX+2][noY];
44             }
45             }else{
46                 int m = (lY+rY)/2;
47
48                 buildY(noX, lX, rX, 2*noY+1, lY, m, v);
49                 buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);
50
51                 seg[noX][noY] = seg[noX][2*noY+1] + seg[
noX][2*noY+2];
52             }
53         }
54
55     void buildX(int noX, int lX, int rX, vector<
vector<int>> &v){
56         if(lX != rX){
57             int m = (lX+rX)/2;
58
59             buildX(2*noX+1, lX, m, v);
60             buildX(2*noX+2, m+1, rX, v);
61         }
62
63         buildY(noX, lX, rX, 0, 0, M - 1, v);
64     }
65
66     void updateY(int noX, int lX, int rX, int noY,
int lY, int rY, int y){
67         if(lY == rY){
68             if(lX == rX){
69                 seg[noX][noY] = !seg[noX][noY];
70             }else{
71                 seg[noX][noY] = seg[2*noX+1][noY] +
seg[2*noX+2][noY];
72             }
73             }else{
74                 int m = (lY+rY)/2;
75
76                 if(y <= m){
77                     updateY(noX, lX, rX, 2*noY+1, lY, m, y
);
78                 }else if(m < y){
79                     updateY(noX, lX, rX, 2*noY+2, m+1, rY
, y);
80                 }
81
82                 seg[noX][noY] = seg[noX][2*noY+1] + seg[
noX][2*noY+2];
83             }
84         }
85
86     void updateX(int noX, int lX, int rX, int x, int

```

```

y){
87     int m = (lX+rX)/2;
88
89     if(lX != rX){
90         if(x <= m){
91             updateX(2*noX+1, lX, m, x, y);
92         }else if(m < x){
93             updateX(2*noX+2, m+1, rX, x, y);
94         }
95     }
96
97     updateY(noX, lX, rX, 0, 0, M - 1, y);
98 }
99
100 int queryY(int noX, int noY, int lY, int rY, int
aY, int bY){
101     if(aY <= lY && rY <= bY) return seg[noX][noY
];
102
103     int m = (lY+rY)/2;
104
105     if(bY <= m) return queryY(noX, 2*noY+1, lY, m
, aY, bY);
106     if(m < aY) return queryY(noX, 2*noY+2, m+1,
rY, aY, bY);
107
108     return queryY(noX, 2*noY+1, lY, m, aY, bY) +
queryY(noX, 2*noY+2, m+1, rY, aY, bY);
109 }
110
111 int queryX(int noX, int lX, int rX, int aX, int
bX, int aY, int bY){
112     if(aX <= lX && rX <= bX) return queryY(noX,
0, 0, M - 1, aY, bY);
113
114     int m = (lX+rX)/2;
115
116     if(bX <= m) return queryX(2*noX+1, lX, m, aX,
bX, aY, bY);
117     if(m < aX) return queryX(2*noX+2, m+1, rX, aX
, bX, aY, bY);
118
119     return queryX(2*noX+1, lX, m, aX, bX, aY, bY)
+ queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
120 }
121
122 void build(vector<vector<int>> &v) {
123     buildX(0, 0, N - 1, v);
124 }
125
126 int query(int aX, int bX, int aY, int bY) {
127     return queryX(0, 0, N - 1, aX, bX, aY, bY);
128 }
129
130 void update(int x, int y) {
131     updateX(0, 0, N - 1, x, y);
132 }
133 };

```

4 Strings

4.1 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],
LCS_table[i][j - 1]);
22         }
23     }
24
25     int index = LCS_table[m][n];
26     char lcsAlgo[index + 1];
27     lcsAlgo[index] = '\0';
28
29     int i = m, j = n;
30     while (i > 0 && j > 0) {
31         if (s1[i - 1] == s2[j - 1]) {
32             lcsAlgo[index - 1] = s1[i - 1];
33             i--;
34             j--;
35             index--;
36         }
37         else if (LCS_table[i - 1][j] > LCS_table[i][j -
1])
38             i--;
39         else
40             j--;
41     }
42
43     return lcsAlgo;
44 }

```

4.2 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 }

```

4.3 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     }
42     return t[v].link;
43 }
44
45 int go(int v, char ch) {
46     int c = ch - 'a';
47     if (t[v].go[c] == -1) {
48         if (t[v].next[c] != -1)
49             t[v].go[c] = t[v].next[c];
50         else
51             t[v].go[c] = v == 0 ? 0 : go(get_link(v),
52     ch);
53     }
54     return t[v].go[c];
55 }

```

4.4 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 }

```

4.5 Generate All Sequences Length K

```

1 // gera todas as possíveis sequências usando as letras
2 // em set (de comprimento n) e que tenham tamanho k
3 // sequence = ""
4 vector<string> generate_sequences(char set[], string
5     sequence, int n, int k) {
6     if (k == 0) {
7         return { sequence };
8     }
9     vector<string> ans;
10    for (int i = 0; i < n; i++) {

```

```

10        auto aux = generate_sequences(set, sequence +
11            set[i], n, k - 1);
12        ans.insert(ans.end(), aux.begin(), aux.end());
13    }
14    // for (auto e : aux) ans.push_back(e);
15    return ans;
16 }

```

4.6 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 }

```

5 Algorithms

5.1 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/abc306\_e
4 // Complexity: O(log n)
5
6 struct SetSum {
7     ll s = 0;
8     multiset<ll> mt;
9     void add(ll x){
10         mt.insert(x);
11         s += x;
12     }
13     int pop(ll x){
14         auto f = mt.find(x);
15         if(f == mt.end()) return 0;
16         mt.erase(f);
17         s -= x;
18         return 1;
19     }
20 };
21
22 struct BigK {
23     int k;
24     SetSum gt, mt;
25     BigK(int _k){
26         k = _k;
27     }
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()));

```

```

39         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 }

```

5.2 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;
13 }

```

5.3 Ternary Search

```

1 double ternary_search(double l, double r) {
2     double eps = 1e-9;           //set the error
3     //limit here
4     while (r - l > eps) {
5         double m1 = l + (r - l) / 3;
6         double m2 = r - (r - l) / 3;
7         double f1 = f(m1);        //evaluates the
8         function at m1
9         double f2 = f(m2);        //evaluates the
10        function at m2
11        if (f1 < f2)
12            l = m1;
13        else
14            r = m2;
15    }
16 }

```

```

13     return f(l);                 //return the
14 }                                maximum of f(x) in [l, r]

```

5.4 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 }

```

5.5 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 }

```

5.6 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;
13 }

```


6 Math

6.1 Ceil

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

6.2 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 }
```

6.3 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 }
```

6.4 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 }
```

6.5 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
10 // matrix 2 x 1 so that
11 // n + 1 = transition * n
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
18 // distinct colors so that
19 // no adjacent place has the same color. In how many
20 // different ways can you do that?
21 // There are 6 ways for the first column to be
22 // colored using 3 distinct colors
23 // ans 6 ways using 2 equal colors and 1 distinct one
24
25 // Adding another column, there are:
26 // 3 ways to go from 2 equal to 2 equal
27 // 2 ways to go from 2 equal to 3 distinct
28 // 2 ways to go from 3 distinct to 2 equal
29 // 2 ways to go from 3 distinct to 3 distinct
30
31 // So we star with matrix 6 6 and multiply it by the
32 // transition 3 2 and get 18 12
33
34 //
35 //      2 2      6 6
36 //      12 12
37 // the we can exponentiate this matrix to find the
38 // nth column
39
40 // Problem:
41 // https://cses.fi/problemset/task/1722/
42
43 // Complexity:
44 // O(log n)
45
46 // How to use:
47 // vector<vector<ll>> v = {{1, 1}, {1, 0}};
48 // Matriz transition = Matriz(v);
49 // cout << fexp(transition, n)[0][1] << '\n';
50
51 using ll = long long;
52
53 const int MOD = 1e9+7;
54
55 struct Matriz{
56     vector<vector<ll>> mat;
57     int rows, columns;
58
59     vector<ll> operator[](int i){
60         return mat[i];
61     }
62
63     Matriz(vector<vector<ll>>& matriz){
64         mat = matriz;
65         rows = mat.size();
66         columns = mat[0].size();
67     }
68
69     Matriz(int row, int column, bool identity=false){
70         rows = row; columns = column;
71         mat.assign(rows, vector<ll>(columns, 0));
72         if(identity) {
73             for(int i = 0; i < min(rows, columns); i
74             ++){
75                 mat[i][i] = 1;
76             }
77         }
78     }
79
80     Matriz operator * (Matriz a) {
81         assert(columns == a.rows);
82         vector<vector<ll>> resp(rows, vector<ll>(a.
83         columns, 0));
84
85         for(int i = 0; i < rows; i++){
86             for(int j = 0; j < a.columns; j++){
87                 for(int k = 0; k < a.rows; k++){
```

```

78         resp[i][j] = (resp[i][j] + (mat[i][k] * 1LL * a[k][j]) % MOD) % MOD;
79     }
80 }
81 }
82 return Matriz(resp);
83 }
84
85 Matriz operator + (Matriz a) {
86     assert(rows == a.rows && columns == a.columns);
87     vector<vector<ll>> resp(rows, vector<ll>(columns, 0));
88     for(int i = 0; i < rows; i++){
89         for(int j = 0; j < columns; j++){
90             resp[i][j] = (resp[i][j] + mat[i][j] + a[i][j]) % MOD;
91         }
92     }
93     return Matriz(resp);
94 }
95 };
96
97 Matriz fexp(Matriz base, ll exponent){
98     Matriz result = Matriz(base.rows, base.columns, 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 }

```

6.6 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 }

```

6.7 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 }

```

6.8 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 }

```

6.9 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 }

```

6.10 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 }

```

6.11 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) {

```

```

4 //      if (c != 0) ans = 0;
5 //      else ans = (x2 - x1 + 1) * (y2 - y1 + 1);
6 // }
7 // else if (a == 0) {
8 //      if (c % b == 0 && y1 <= c / b && y2 >= c / b)
9 //          ans = (x2 - x1 + 1);
10 //      else ans = 0;
11 // }
12 // else if (b == 0) {
13 //      if (c % a == 0 && x1 <= c / a && x2 >= c / a)
14 //          ans = (y2 - y1 + 1);
15 //      else ans = 0;
16 // }
17 // Careful when a or b are negative or zero
18 // if (ans == -1) ans = find_all_solutions(a, b, c,
19 //      x1, x2, y1, y2);
20 // cout << ans << '\n';
21 // Problems:
22 // https://www.spoj.com/problems/CEQU/
23 // http://codeforces.com/problemsets/acmsguru/problem
24 // 99999/106
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 bool find_any_solution(int a, int b, int c, int &x0,
42     int &y0, int &g) {
43     g = gcd(abs(a), abs(b), x0, y0);
44     if (c % g) {
45         return false;
46     }
47     x0 *= c / g;
48     y0 *= c / g;
49     if (a < 0) x0 = -x0;
50     if (b < 0) y0 = -y0;
51     return true;
52 }
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     int sign_a = a > 0 ? +1 : -1;
68     int sign_b = b > 0 ? +1 : -1;

```

```

69
70     shift_solution(x, y, a, b, (minx - x) / b);
71     if (x < minx)
72         shift_solution(x, y, a, b, sign_b);
73     if (x > maxx)
74         return 0;
75     int lx1 = x;
76
77     shift_solution(x, y, a, b, (maxx - x) / b);
78     if (x > maxx)
79         shift_solution(x, y, a, b, -sign_b);
80     int rx1 = x;
81
82     shift_solution(x, y, a, b, -(miny - y) / a);
83     if (y < miny)
84         shift_solution(x, y, a, b, -sign_a);
85     if (y > maxy)
86         return 0;
87     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 }

```

7 Misc

7.1 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 }

```

7.2 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 }

```

8 Template

8.1 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 int main() {

```

```

17     ios::sync_with_stdio(false);
18     cin.tie(NULL);
19
20
21
22     return 0;
23 }

```

8.2 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 #define vi vector<int>
7 #define ll long long
8 #define pb push_back
9 #define mp make_pair
10 #define ff first
11 #define ss second
12 #define pii pair<int, int>
13 #define MOD 1000000007
14 #define sqr(x) ((x) * (x))
15 #define all(x) (x).begin(), (x).end()
16 #define FOR(i, j, n) for (int i = j; i < n; i++)
17 #define qle(i, n) (i == n ? "\n" : " ")
18 #define endl "\n"
19 const int oo = 1e9;
20 const int MAX = 1e6;
21
22 int32_t main(){ optimize;
23
24     return 0;
25 }

```