



# Notebook - Maratona de Programação

## Lenhadoras de Segtree

### Contents

<b>1 DP</b>	<b>2</b>	<b>3.9 Lazy</b>	<b>15</b>
1.1 Substr Palindrome	2	3.10 Lazy Dynamic Implicit Sparse	16
1.2 Coins	2	3.11 Segtree2d	17
1.3 Minimum Coin Change	2	<b>4 Strings</b>	<b>18</b>
1.4 Edit Distance	2	4.1 Lcs	18
1.5 Digits	2	4.2 Kmp	19
1.6 Knapsack With Index	3	4.3 Z-function	19
1.7 Kadane	3	4.4 Generate All Sequences Length K	19
1.8 Knapsack	3	4.5 Generate All Permutations	19
<b>2 Graphs</b>	<b>4</b>	<b>5 Algorithms</b>	<b>19</b>
2.1 Kruskal	4	5.1 Binary Search Last True	19
2.2 Prim	4	5.2 Ternary Search	19
2.3 Dijkstra	4	5.3 Delta-encoding	19
2.4 Bellman Ford	5	5.4 Lis	20
2.5 Bipartite	5	5.5 Binary Search First True	20
2.6 Floyd Warshall	5	<b>6 Math</b>	<b>20</b>
2.7 Tree Diameter	5	6.1 Ceil	20
2.8 Centroid Decomposition	6	6.2 Sieve Of Eratosthenes	20
2.9 Lca	6	6.3 Crt	20
2.10 Ford Fulkerson Edmonds Karp	7	6.4 Check If Bit Is On	20
2.11 Dinic	7	6.5 Matrix Exponentiation	20
2.12 Find Cycle	8	6.6 Fast Exponentiation	21
2.13 Tarjan Bridge	8	6.7 Divisors	21
2.14 Cycle Path Recovery	9	6.8 Binary To Decimal	21
2.15 Centroid Find	9	6.9 Multiplicative Inverse	21
2.16 Small To Large	9	6.10 Prime Factors	22
<b>3 Data Structures</b>	<b>10</b>	6.11 Linear Diophantine Equation	22
3.1 Ordered Set	10	<b>7 Misc</b>	<b>23</b>
3.2 Priority Queue	10	7.1 Split	23
3.3 Dsu	10	7.2 Int128	23
3.4 Persistent	11	<b>8 Template</b>	<b>23</b>
3.5 Minimum And Amount	12	8.1 Template Clean	23
3.6 Range Query Point Update	12	8.2 Template	23
3.7 Segment With Maximum Sum	13		
3.8 Dynamic Implicit Sparse	14		

# 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 vector<int> parent, rank;
2
3 void make_set(int v) {
4     parent[v] = v;
5     rank[v] = 0;
6 }
7
8 int find_set(int v) {
9     if (v == parent[v])
10         return v;
11     return parent[v] = find_set(parent[v]);
12 }
13
14 void union_sets(int a, int b) {
15     a = find_set(a);
16     b = find_set(b);
17     if (a != b) {
18         if (rank[a] < rank[b])
19             swap(a, b);
20         parent[b] = a;
21         if (rank[a] == rank[b])
22             rank[a]++;
23     }
24 }
25
26 struct Edge {
27     int u, v, weight;
28     bool operator< (const Edge & other) {
29         return weight < other.weight;
30     }
31 };
32
33 int n;
34 vector<Edge> edges;
35
36 int cost = 0;
37 vector<Edge> result;
38 parent.resize(n);
39 rank.resize(n);
40 for (int i = 0; i < n; i++)
41     make_set(i);
42
43 sort(edges.begin(), edges.end());
44
45 for (Edge e : edges) {
46     if (find_set(e.u) != find_set(e.v)) {
47         cost += e.weight;
48         result.push_back(e);
49         union_sets(e.u, e.v);
50     }
51 }
```

### 2.2 Prim

```
1 int n;
2 vector<vector<int>> adj; // adjacency matrix of graph
3 const int INF = 1000000000; // weight INF means there
   is no edge
4
5 struct Edge {
6     int w = INF, to = -1;
7 };
8
9 void prim() {
10     int total_weight = 0;
11     vector<bool> selected(n, false);
12     vector<Edge> min_e(n);
13     min_e[0].w = 0;
```

```
14
15     for (int i=0; i<n; ++i) {
16         int v = -1;
17         for (int j = 0; j < n; ++j) {
18             if (!selected[j] && (v == -1 || min_e[j].
19                 w < min_e[v].w))
20                 v = j;
21         }
22         if (min_e[v].w == INF) {
23             cout << "No MST!" << endl;
24             exit(0);
25         }
26         selected[v] = true;
27         total_weight += min_e[v].w;
28         if (min_e[v].to != -1)
29             cout << v << " " << min_e[v].to << endl;
30
31         for (int to = 0; to < n; ++to) {
32             if (adj[v][to] < min_e[to].w)
33                 min_e[to] = {adj[v][to], v};
34         }
35     }
36 }
37
38     cout << total_weight << endl;
39 }
```

### 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     int n = adj.size();
7     d.assign(n, INF);
8     p.assign(n, -1);
9
10    d[s] = 0;
11    set<pair<int, int>> q;
12    q.insert({0, s});
13    while (!q.empty()) {
14        int v = q.begin()->second;
15        q.erase(q.begin());
16
17        for (auto edge : adj[v]) {
18            int to = edge.first;
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
56     bool sptSet[V];
57     for (int i = 0; i < V; i++)
58         dist[i] = INT_MAX, sptSet[i] = false;
59
60     dist[src] = 0;
61
62     for (int count = 0; count < V - 1; count++) {
63         int u = minDistance(dist, sptSet, V);
64
65         sptSet[u] = true;
66
67         for (int v = 0; v < V; v++)
68             if (!sptSet[v] && adj[u][v]
69                 && dist[u] != INT_MAX
70                 && dist[u] + adj[u][v] < dist[v])
71                 dist[v] = dist[u] + adj[u][v];
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
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 }

```

## 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){
27                     dist[i][j] = min(dist[i][j], dist
28                         [i][k] + dist[k][j]);
29                 }
30             }
31         }
32     }
33 }

```

## 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
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 }

```

## 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
  the following formula
16 // dist[a] + dist[b] - 2*dist[lca(a, b)]
17
18 const int MAX = 2e5+17;
19
20 const int BITS = 32;
21
22 vector<int> adj[MAX];
23 // vector<pair<int, int>> adj[MAX];
24 // int dist[MAX];
25
26 int timer;
27 vector<int> tin, tout;
28 vector<vector<int>> up;
29
30 void dfs(int v, int p)
31 {
32     tin[v] = ++timer;
33     up[v][0] = p;
34
35     for (int i = 1; i <= BITS; ++i) {

```

```

36         up[v][i] = up[up[v][i-1]][i-1];
37     }
38
39     for (auto u : adj[v]) {
40         if (u != p) {
41             dfs(u, v);
42         }
43     }
44
45     /*for (auto [u, peso] : adj[v]) {
46         if (u != p) {
47             dist[u] = dist[v] + peso;
48             dfs(u, v);
49         }
50     }*/
51
52     tout[v] = ++timer;
53 }
54
55 bool is_ancestor(int u, int v)
56 {
57     return tin[u] <= tin[v] && tout[u] >= tout[v];
58 }
59
60 int lca(int u, int v)
61 {
62     if (is_ancestor(u, v))
63         return u;
64     if (is_ancestor(v, u))
65         return v;
66     for (int i = BITS; i >= 0; --i) {
67         if (!is_ancestor(up[u][i], v))
68             u = up[u][i];
69     }
70     return up[u][0];
71 }
72
73 void preprocess(int root) {
74     tin.resize(MAX);
75     tout.resize(MAX);
76     timer = 0;
77     up.assign(MAX, vector<int>(BITS + 1));
78     dfs(root, root);
79 }

```

## 2.10 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 }

```

## 2.11 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.flow() << '\n';
22
23 struct FlowEdge {
24     int v, u;
25     long long cap, flow = 0;
26     FlowEdge(int v, int u, long long cap) : v(v), u(u)
, cap(cap) {}

```

```

27 };
28
29 struct Dinic {
30     const long long flow_inf = 1e18;
31     vector<FlowEdge> edges;
32     vector<vector<int>> adj;
33     int n, m = 0;
34     int s, t;
35     vector<int> level, ptr;
36     queue<int> q;
37
38     Dinic(int n, int s, int t) : n(n), s(s), t(t) {
39         adj.resize(n);
40         level.resize(n);
41         ptr.resize(n);
42     }
43
44     void add_edge(int v, int u, long long cap) {
45         edges.emplace_back(v, u, cap);
46         edges.emplace_back(u, v, 0);
47         adj[v].push_back(m);
48         adj[u].push_back(m + 1);
49         m += 2;
50     }
51
52     bool bfs() {
53         while (!q.empty()) {
54             int v = q.front();
55             q.pop();
56             for (int id : adj[v]) {
57                 if (edges[id].cap - edges[id].flow <
58                     1)
59                     continue;
60                 if (level[edges[id].u] != -1)
61                     continue;
62                 level[edges[id].u] = level[v] + 1;
63                 q.push(edges[id].u);
64             }
65             return level[t] != -1;
66         }
67
68         long long dfs(int v, long long pushed) {
69             if (pushed == 0)
70                 return 0;
71             if (v == t)
72                 return pushed;
73             for (int& cid = ptr[v]; cid < (int)adj[v].
74                 size(); cid++) {
75                 int id = adj[v][cid];
76                 int u = edges[id].u;
77                 if (level[v] + 1 != level[u] || edges[id]
78                     .cap - edges[id].flow < 1)
79                     continue;
80                 long long tr = dfs(u, min(pushed, edges[
81                     id].cap - edges[id].flow));
82                 if (tr == 0)
83                     continue;
84                 edges[id].flow += tr;
85                 edges[id ^ 1].flow -= tr;
86                 return tr;
87             }
88             return 0;
89         }
90
91         long long flow() {
92             long long f = 0;
93             while (true) {
94                 fill(level.begin(), level.end(), -1);
95                 level[s] = 0;
96                 q.push(s);
97                 if (!bfs())
98                     break;
99             }
100         }
101     };

```

```

96         fill(ptr.begin(), ptr.end(), 0);
97         while (long long pushed = dfs(s, flow_inf
98             )) {
99             f += pushed;
100         }
101         return f;
102     }
103 };

```

## 2.12 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 }

```

## 2.13 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;
15
16 vector<bool> visited;
17 vector<int> tin, low;
18 int timer;
19
20 void dfs(int v, int p) {
21     visited[v] = true;
22     tin[v] = low[v] = timer++;
23     for (int u : adj[v]) {
24         if (u == p) continue;
25         if (!visited[u]) {
26             dfs(u, v);
27             low[v] = min(low[v], low[u]);
28             if (low[u] > tin[v]) {
29                 // Bridge found
30             }
31         } else {
32             low[v] = min(low[v], low[u]);
33         }
34     }
35 }

```



```

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 }

```

## 2.14 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 }

```

## 2.15 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 }

```

## 2.16 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 ==
22                 vmax[curr]) {
23                 sum_num[curr] += item;
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
5 // ordered set
6 // order_of_key(k) - returns number of elements
7 // strictly smaller than k
8 // find_by_order(n) - return an iterator pointing to
9 // the k-th element in the ordered set (counting
10 // from zero).
11
12 // 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
22 // ordered_set declaration
23 // The ordered_set becomes an ordered_multiset
24 // So the set can contain elements that are equal
25
26 #include <ext/pb_ds/assoc_container.hpp>
27 #include <ext/pb_ds/tree_policy.hpp>
28
29 using namespace __gnu_pbds;
30 template <typename T>
31 using ordered_set = tree<T, null_type, less<T>,
32     rb_tree_tag, tree_order_statistics_node_update>;

```

### 3.2 Priority Queue

```

1 // Description:
2 // Keeps the largest (by default) element at the top
3 // of the queue
4
5 // Problem:
6 // https://cses.fi/problemset/task/1164/
7
8 // Complexity:
9 // O(log n) for push and pop
10 // O(1) for looking at the element at the top
11
12 // How to use:
13 // priority_queue<int> pq;
14 // pq.push(1);
15 // pq.top();
16 // pq.pop()
17
18 // Notes
19 // To use the priority queue keeping the smallest
20 // element at the top
21
22 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
3 // different versions of the segtree between each
4 // update
5 // Indexed at one
6 // Query - get sum of elements from range (l, r)
7 // inclusive
8 // Update - update element at position id to a value
9 // val
10 // Problem:
11 // https://cses.fi/problemset/task/1737/
12 // Complexity:
13 // O(log n) for both query and update

```

```

12
13 // How to use:
14 // vector<int> raiz(MAX); // vector to store the
15 // roots of each version
16 // Segtree seg = Segtree(INF);
17 // raiz[0] = seg.create(); // null node
18 // curr = 1; // keep track of the last version
19 // raiz[k] = seg.update(raiz[k], idx, val); //
20 // updating version k
21 // seg.query(raiz[k], l, r) // querying version k
22 // raiz[++curr] = raiz[k]; // create a new version
23 // based on version k
24
25 const int MAX = 2e5+17;
26 const int INF = 1e9+17;
27
28 typedef long long ftype;
29
30 struct Segtree {
31     vector<ftype> seg, d, e;
32     const ftype NEUTRAL = 0;
33     int n;
34
35     Segtree(int n) {
36         this->n = n;
37     }
38
39     ftype f(ftype a, ftype b) {
40         return a + b;
41     }
42
43     ftype create() {
44         seg.push_back(0);
45         e.push_back(0);
46         d.push_back(0);
47         return seg.size() - 1;
48     }
49
50     ftype query(int pos, int ini, int fim, int p, int
51     q) {
52         if (q < ini || p > fim) return NEUTRAL;
53         if (pos == 0) return 0;
54         if (p <= ini && fim <= q) return seg[pos];
55         int m = (ini + fim) >> 1;
56         return f(query(e[pos], ini, m, p, q), query(d
57         [pos], m + 1, fim, p, q));
58     }
59
60     int update(int pos, int ini, int fim, int id, int
61     val) {
62         int novo = create();
63
64         seg[novo] = seg[pos];
65         e[novo] = e[pos];
66         d[novo] = d[pos];
67
68         if (ini == fim) {
69             seg[novo] = val;
70             return novo;
71         }
72
73         int m = (ini + fim) >> 1;
74
75         if (id <= m) e[novo] = update(e[novo], ini, m
76         , id, val);
77         else d[novo] = update(d[novo], m + 1, fim, id
78         , val);
79
80         seg[novo] = f(seg[e[novo]], seg[d[novo]]);
81
82         return novo;
83     }
84 }

```

```

77
78     ftype query(int pos, int p, int q) {
79         return query(pos, 1, n, p, q);
80     }
81
82     int update(int pos, int id, int val) {
83         return update(pos, 1, n, id, val);
84     }
85 };

```

### 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;

```

```

57
58         return f(query(e, ini, m, p, q), query(d, m +
59         1, fim, p, q));
60     }
61
62     void update(int pos, int ini, int fim, int id,
63     int val) {
64         if (ini > id || fim < id) {
65             return;
66         }
67
68         if (ini == id && fim == id) {
69             seg[pos] = mp(val, 1);
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     }
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) {
116             cout << e.ff << ' ' << e.ss << '\n';
117         }
118         cout << '\n';
119     }
120 };

```

### 3.6 Range Query Point Update

```

1 // Description:
2 // Indexed at zero
3 // Query - get sum of elements from range (l, r)
   inclusive

```

```

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/B
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 // Notes
17 // Change neutral element and f function to perform a
  different operation
18
19 // If you want to change the operations to point
  query and range update
20 // Use the same segtree, but perform the following
  operations
21 // Query - seg.query(0, id);
22 // Update - seg.update(1, v); seg.update(r + 1, -v);
23
24 typedef long long ftype;
25
26 struct Segtree {
27     vector<ftype> seg;
28     int n;
29     const ftype NEUTRAL = 0;
30
31     Segtree(int n) {
32         int sz = 1;
33         while (sz < n) sz *= 2;
34         this->n = sz;
35
36         seg.assign(2*sz, NEUTRAL);
37     }
38
39     ftype f(ftype a, ftype b) {
40         return a + b;
41     }
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             return;
70         }
71
72         seg[pos] = f(seg[e], seg[d]);
73     }
74
75     void build(int pos, int ini, int fim, vector<int>
76     &v) {
77         if (ini == fim) {
78             if (ini < (int)v.size()) {
79                 seg[pos] = v[ini];
80             }
81             return;
82         }
83
84         build(e, ini, m, v);
85         build(d, m + 1, fim, v);
86
87         seg[pos] = f(seg[e], seg[d]);
88     }
89
90     ftype query(int p, int q) {
91         return query(0, 0, n - 1, p, q);
92     }
93
94     void update(int id, int val) {
95         update(0, 0, n - 1, id, val);
96     }
97
98     void build(vector<int> &v) {
99         build(0, 0, n - 1, v);
100     }
101
102     void debug() {
103         for (auto e : seg) {
104             cout << e << ' ';
105         }
106         cout << '\n';
107     }
108 };

```

### 3.7 Segment With Maximum Sum

```

1 // Description:
2 // Query - get sum of segment that is maximum among
  all segments
3 // E.g
4 // Array: 5 -4 4 3 -5
5 // Maximum segment sum: 8 because 5 + (-4) + 4 = 8
6 // Update - update element at position id to a value
  val
7
8 // Problem:
9 // https://codeforces.com/edu/course/2/lesson/4/2/
  practice/contest/273278/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 // The maximum segment sum can be a negative number
20 // In that case, taking zero elements is the best
   choice
21 // So we need to take the maximum between 0 and the
   query
22 // max(OLL, seg.query(0, n).max_seg)
23
24 using ll = long long;
25
26 typedef ll ftype_node;
27
28 struct Node {
29     ftype_node max_seg;
30     ftype_node pref;
31     ftype_node suf;
32     ftype_node sum;
33
34     Node(ftype_node max_seg, ftype_node pref,
          ftype_node suf, ftype_node sum) : max_seg(max_seg
          ), pref(pref), suf(suf), sum(sum) {};
35 };
36
37 typedef Node ftype;
38
39 struct Segtree {
40     vector<ftype> seg;
41     int n;
42     const ftype NEUTRAL = Node(0, 0, 0, 0);
43
44     Segtree(int n) {
45         int sz = 1;
46         // potencia de dois mais proxima
47         while (sz < n) sz *= 2;
48         this->n = sz;
49
50         // numero de nos da seg
51         seg.assign(2*sz, NEUTRAL);
52     }
53
54     ftype f(ftype a, ftype b) {
55         ftype_node max_seg = max({a.max_seg, b.
max_seg, a.suf + b.pref});
56         ftype_node pref = max(a.pref, a.sum + b.pref);
57
58         ftype_node suf = max(b.suf, b.sum + a.suf);
59         ftype_node sum = a.sum + b.sum;
60
61         return Node(max_seg, pref, suf, sum);
62     }
63
64     ftype query(int pos, int ini, int fim, int p, int
q) {
65         if (ini >= p && fim <= q) {
66             return seg[pos];
67         }
68
69         if (q < ini || p > fim) {
70             return NEUTRAL;
71         }
72
73         int e = 2*pos + 1;
74         int d = 2*pos + 2;
75         int m = ini + (fim - ini) / 2;
76
77         return f(query(e, ini, m, p, q), query(d, m +
1, fim, p, q));
78
79 void update(int pos, int ini, int fim, int id,
int val) {
80     if (ini > id || fim < id) {
81         return;
82     }
83
84     if (ini == id && fim == id) {
85         seg[pos] = Node(val, val, val, val);
86
87         return;
88     }
89
90     int e = 2*pos + 1;
91     int d = 2*pos + 2;
92     int m = ini + (fim - ini) / 2;
93
94     update(e, ini, m, id, val);
95     update(d, m + 1, fim, id, val);
96
97     seg[pos] = f(seg[e], seg[d]);
98 }
99
100 void build(int pos, int ini, int fim, vector<int>
&v) {
101     if (ini == fim) {
102         // se a posição existir no array original
103         // seg tamanho potencia de dois
104         if (ini < (int)v.size()) {
105             seg[pos] = Node(v[ini], v[ini], v[ini]
], v[ini]);
106         }
107         return;
108     }
109
110     int e = 2*pos + 1;
111     int d = 2*pos + 2;
112     int m = ini + (fim - ini) / 2;
113
114     build(e, ini, m, v);
115     build(d, m + 1, fim, v);
116
117     seg[pos] = f(seg[e], seg[d]);
118 }
119
120 ftype query(int p, int q) {
121     return query(0, 0, n - 1, p, q);
122 }
123
124 void update(int id, int val) {
125     update(0, 0, n - 1, id, val);
126 }
127
128 void build(vector<int> &v) {
129     build(0, 0, n - 1, v);
130 }
131
132 void debug() {
133     for (auto e : seg) {
134         cout << e.max_seg << ' ' << e.pref << ' '
<< e.suf << ' ' << e.sum << '\n';
135     }
136     cout << '\n';
137 }
138 };

```

### 3.8 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://cses.fi/problemset/task/1648
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 struct Segtree {
32     vector<ftype> seg, d, e, lazy;
33     const ftype NEUTRAL = 0;
34     int n;
35
36     Segtree(int n) {
37         this->n = n;
38     }
39
40     ftype f(ftype a, ftype b) {
41         return a + b;
42     }
43
44     ftype create() {
45         seg.push_back(0);
46         e.push_back(0);
47         d.push_back(0);
48         return seg.size() - 1;
49     }
50
51     ftype query(int pos, int ini, int fim, int p, int
        q) {
52         if (q < ini || p > fim) return NEUTRAL;
53         if (pos == 0) return 0;
54         if (p <= ini && fim <= q) return seg[pos];
55         int m = (ini + fim) >> 1;
56         return f(query(e[pos], ini, m, p, q), query(d
            [pos], m + 1, fim, p, q));
57     }
58
59     void update(int pos, int ini, int fim, int id,
        int val) {
60         if (ini > id || fim < id) {
61             return;
62         }
63
64         if (ini == fim) {
65             seg[pos] = val;
66
67             return;
68         }
69
70         int m = (ini + fim) >> 1;
71
72         if (id <= m) {
73             if (e[pos] == 0) e[pos] = create();
74             update(e[pos], ini, m, id, val);
75         } else {

```

```

76         if (d[pos] == 0) d[pos] = create();
77         update(d[pos], m + 1, fim, id, val);
78     }
79
80     seg[pos] = f(seg[e[pos]], seg[d[pos]]);
81 }
82
83 ftype query(int p, int q) {
84     return query(1, 1, n, p, q);
85 }
86
87 void update(int id, int val) {
88     update(1, 1, n, id, val);
89 }
90 };

```

### 3.9 Lazy

```

1 // Description:
2 // Query - get sum of elements from range (l, r)
    inclusive
3 // Update - add a value val to elements from range (
    l, r) inclusive
4
5 // Problem:
6 // https://codeforces.com/edu/course/2/lesson/5/1/
    practice/contest/279634/problem/A
7
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
    different operation
17
18 typedef long long ftype;
19
20 struct Segtree {
21     vector<ftype> seg;
22     vector<ftype> lazy;
23     int n;
24     const ftype NEUTRAL = 0;
25     const ftype NEUTRAL_LAZY = -1;
26
27     Segtree(int n) {
28         int sz = 1;
29         while (sz < n) sz *= 2;
30         this->n = sz;
31
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); 117
53
54     seg[e] = apply_lazy(seg[e], lazy[pos], m - 118
ini + 1); 119
55     seg[d] = apply_lazy(seg[d], lazy[pos], fim - 120
m); 121
56
57     lazy[pos] = NEUTRAL_LAZY; 122
58 } 123
59
60 ftype f(ftype a, ftype b) { 124
61     return a + b; 125
62 } 126
63
64 ftype query(int pos, int ini, int fim, int p, int 127
q) { 128
65     propagate(pos, ini, fim); 129
66
67     if (ini >= p && fim <= q) { 130
68         return seg[pos]; 131
69     } 132
70
71     if (q < ini || p > fim) { 133
72         return NEUTRAL; 134
73     } 135
74
75     int e = 2*pos + 1; 136
76     int d = 2*pos + 2; 137
77     int m = ini + (fim - ini) / 2; 138
78
79     return f(query(e, ini, m, p, q), query(d, m + 147
1, fim, p, q));
80 }
81
82 void update(int pos, int ini, int fim, int p, int 1
q, int val) { 2
83     propagate(pos, ini, fim); 3
84
85     if (ini > q || fim < p) { 4
86         return; 5
87     } 6
88
89     if (ini >= p && fim <= q) { 7
90         lazy[pos] = apply_lazy(lazy[pos], val, 1) 8
; 9
91         seg[pos] = apply_lazy(seg[pos], val, fim 10
- ini + 1); 11
92
93         return; 12
94     } 13
95
96     int e = 2*pos + 1; 14
97     int d = 2*pos + 2; 15
98     int m = ini + (fim - ini) / 2; 16
99
100     update(e, ini, m, p, q, val); 17
101     update(d, m + 1, fim, p, q, val); 18
102
103     seg[pos] = f(seg[e], seg[d]); 19
104 } 20
105
106 void build(int pos, int ini, int fim, vector<int> 21
&v) { 22
107     if (ini == fim) { 23
108         if (ini < (int)v.size()) { 24
109             seg[pos] = v[ini]; 25
110         } 26
111         return; 27
112     } 28
113
114     int e = 2*pos + 1; 29
115     int d = 2*pos + 2; 30
116     int m = ini + (fim - ini) / 2; 31

```

```

        build(e, ini, m, v);
        build(d, m + 1, fim, v);

        seg[pos] = f(seg[e], seg[d]);
    }

    ftype query(int p, int q) {
        return query(0, 0, n - 1, p, q);
    }

    void update(int p, int q, int val) {
        update(0, 0, n - 1, p, q, val);
    }

    void build(vector<int> &v) {
        build(0, 0, n - 1, v);
    }

    void debug() {
        for (auto e : seg) {
            cout << e << ' ';
        }
        cout << '\n';
        for (auto e : lazy) {
            cout << e << ' ';
        }
        cout << '\n';
        cout << '\n';
    }
}

```

### 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
;
101             seg[pos] = apply_lazy(seg[pos], val, fim
- ini + 1);
102
103             return;
104         }
105
106         int m = (ini + fim) >> 1;
107
108         if (e[pos] == 0) e[pos] = create();
109         update(e[pos], ini, m, p, q, val);
110
111         if (d[pos] == 0) d[pos] = create();
112         update(d[pos], m + 1, fim, p, q, val);
113
114         seg[pos] = f(seg[e[pos]], seg[d[pos]]);
115     }
116
117     ftype query(int p, int q) {
118         return query(1, 1, n, p, q);
119     }
120
121     void update(int p, int q, int val) {
122         update(1, 1, n, p, q, val);
123     };
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

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
107     if(m < aY) return queryY(noX, 2*noY+2, m+1,
rY, aY, bY);
108
109     return queryY(noX, 2*noY+1, lY, m, aY, bY) +
queryY(noX, 2*noY+2, m+1, rY, aY, bY);
110 }
111
112 int queryX(int noX, int lX, int rX, int aX, int
bX, int aY, int bY){
113     if(aX <= lX && rX <= bX) return queryY(noX,
0, 0, M - 1, aY, bY);
114
115     int m = (lX+rX)/2;
116
117     if(bX <= m) return queryX(2*noX+1, lX, m, aX,
bX, aY, bY);
118
119     if(m < aX) return queryX(2*noX+2, m+1, rX, aX,
bX, aY, bY);
120
121     return queryX(2*noX+1, lX, m, aX, bX, aY, bY)
+ queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
122 }
123
124 void build(vector<vector<int>> &v) {
125     buildX(0, 0, N - 1, v);
126 }
127
128 int query(int aX, int bX, int aY, int bY) {
129     return queryX(0, 0, N - 1, aX, bX, aY, bY);
130 }
131
132 void update(int x, int y) {
133     updateX(0, 0, N - 1, x, y);
134 }
135 };

```

## 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';

```

```

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 - 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 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 }

```

## 4.4 Generate All Sequences Length K

```

1 // gera todas as ípossveis êsequencias 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 +
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 }

```

```

14
15     return ans;
16 }

```

## 4.5 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 }

```

# 5 Algorithms

## 5.1 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 }

```

## 5.2 Ternary Search

```

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

```

## 5.3 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    }
14 }

```

```

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 }

```

## 5.4 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 }

```

## 5.5 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;
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 int n;
2 vector<bool> is_prime(n+1, true);
3 is_prime[0] = is_prime[1] = false;
4 for (int i = 2; i <= n; i++) {
5     if (is_prime[i] && (long long)i * i <= n) {
6         for (int j = i * i; j <= n; j += i)
7             is_prime[j] = false;

```

```

8     }
9 }

```

## 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 //
36 //
37 //
38 //
39 //
40 //
41 //
42 //
43 //
44 //
45 //
46 //
47 //
48 //
49 //
50 //
51 //
52 //
53 //
54 //
55 //
56 //
57 //
58 //
59 //
60 //
61 //
62 //
63 //
64 //
65 //
66 //
67 //
68 //
69 //
70 //
71 //
72 //
73 //
74 //
75 //
76 //
77 //
78 //
79 //
80 //
81 //
82 //
83 //
84 //
85 //
86 //
87 //
88 //
89 //
90 //
91 //
92 //
93 //
94 //
95 //
96 //
97 //
98 //
99 //
100 //

```

```

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     }
100 };

```

```

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 }

```

## 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) {
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';
22 // 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 // 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;
89
90     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      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 }

```