



# Notebook - Maratona de Programação

## Lenhadoras de Segtree

### Contents

<b>1 Misc</b>	<b>2</b>	<b>5.14 Lca</b>	<b>15</b>
1.1 Split	2	<b>5.15 Centroid Decomposition</b>	<b>16</b>
1.2 Int128	2	<b>6 Strings</b>	<b>16</b>
<b>2 Geometry</b>	<b>2</b>	6.1 Kmp	16
<b>3 Data Structures</b>	<b>2</b>	6.2 Lcs	17
3.1 Range Query Point Update	2	6.3 Generate All Permutations	17
3.2 Minimum And Amount	3	6.4 Generate All Sequences Length K	17
3.3 Dynamic Implicit Sparse	4	6.5 Z-function	17
3.4 Lazy Dynamic Implicit Sparse	4	<b>7 DP</b>	<b>17</b>
3.5 Lazy	5	7.1 Knapsack	17
3.6 Segment With Maximum Sum	6	7.2 Substr Palindrome	17
3.7 Segtree2d	8	7.3 Edit Distance	18
3.8 Persistent	9	7.4 Knapsack With Index	18
3.9 Dsu	9	7.5 Minimum Coin Change	18
3.10 Ordered Set	10	7.6 Digits	18
3.11 Priority Queue	10	7.7 Coins	19
<b>4 Template</b>	<b>10</b>	7.8 Kadane	19
4.1 Template	10	<b>8 Math</b>	<b>19</b>
4.2 Template Clean	10	8.1 Multiplicative Inverse	19
<b>5 Graphs</b>	<b>11</b>	8.2 Divisors	19
5.1 Floyd Warshall	11	8.3 Prime Factors	20
5.2 Tree Diameter	11	8.4 Binary To Decimal	20
5.3 Cycle Path Recovery	11	8.5 Sieve Of Eratosthenes	20
5.4 Bipartite	12	8.6 Check If Bit Is On	20
5.5 Find Cycle	12	8.7 Crt	20
5.6 Dinic	12	8.8 Ceil	20
5.7 Bellman Ford	13	8.9 Matrix Exponentiation	20
5.8 Dijkstra	13	8.10 Linear Diophantine Equation	21
5.9 Tarjan Bridge	13	8.11 Fast Exponentiation	22
5.10 Centroid Find	14	<b>9 Algorithms</b>	<b>22</b>
5.11 Small To Large	14	9.1 Lis	22
5.12 Prim	15	9.2 Ternary Search	22
5.13 Kruskall	15	9.3 Binary Search First True	22
		9.4 Delta-encoding	22
		9.5 Binary Search Last True	23

# 1 Misc

## 1.1 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 }
```

## 1.2 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 }
```

# 2 Geometry

# 3 Data Structures

## 3.1 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:
```

```
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
18 // different operation
19
20 // If you want to change the operations to point
21 // query and range update
22 // Use the same segtree, but perform the following
23 // operations
24 // Query - seg.query(0, id);
25 // Update - seg.update(1, v); seg.update(r + 1, -v);
26
27 typedef long long ftype;
28
29 struct Segtree {
30     vector<ftype> seg;
31     int n;
32     const ftype NEUTRAL = 0;
33
34     Segtree(int n) {
35         int sz = 1;
36         while (sz < n) sz *= 2;
37         this->n = sz;
38
39         seg.assign(2*sz, NEUTRAL);
40     }
41
42     ftype f(ftype a, ftype b) {
43         return a + b;
44     }
45
46     ftype query(int pos, int ini, int fim, int p, int
47     q) {
48         if (ini >= p && fim <= q) {
49             return seg[pos];
50         }
51
52         if (q < ini || p > fim) {
53             return NEUTRAL;
54         }
55
56         int e = 2*pos + 1;
57         int d = 2*pos + 2;
58         int m = ini + (fim - ini) / 2;
59
60         return f(query(e, ini, m, p, q), query(d, m +
61         1, fim, p, q));
62     }
63
64     void update(int pos, int ini, int fim, int id,
65     int val) {
66         if (ini > id || fim < id) {
67             return;
68         }
69
70         if (ini == id && fim == id) {
71             seg[pos] = val;
72             return;
73         }
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 }
```

```

77     seg[pos] = f(seg[e], seg[d]);
78 }
79
80 void build(int pos, int ini, int fim, vector<int>
81 &v) {
82     if (ini == fim) {
83         if (ini < (int)v.size()) {
84             seg[pos] = v[ini];
85         }
86         return;
87     }
88     int e = 2*pos + 1;
89     int d = 2*pos + 2;
90     int m = ini + (fim - ini) / 2;
91
92     build(e, ini, m, v);
93     build(d, m + 1, fim, v);
94
95     seg[pos] = f(seg[e], seg[d]);
96 }
97
98 ftype query(int p, int q) {
99     return query(0, 0, n - 1, p, q);
100 }
101
102 void update(int id, int val) {
103     update(0, 0, n - 1, id, val);
104 }
105
106 void build(vector<int> &v) {
107     build(0, 0, n - 1, v);
108 }
109
110 void debug() {
111     for (auto e : seg) {
112         cout << e << ' ';
113     }
114     cout << '\n';
115 }
116 };

```

## 3.2 Minimum And Amount

```

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

```

```

95         build(d, m + 1, fim, v);
96
97         seg[pos] = f(seg[e], seg[d]);
98     }
99
100     ftype query(int p, int q) {
101         return query(0, 0, n - 1, p, q);
102     }
103
104     void update(int id, int val) {
105         update(0, 0, n - 1, id, val);
106     }
107
108     void build(vector<int> &v) {
109         build(0, 0, n - 1, v);
110     }
111
112     void debug() {
113         for (auto e : seg) {
114             cout << e.ff << ' ' << e.ss << '\n';
115         }
116         cout << '\n';
117     }
118 };

```

### 3.3 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) {

```

```

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
52     q) {
53         if (q < ini || p > fim) return NEUTRAL;
54         if (pos == 0) return 0;
55         if (p <= ini && fim <= q) return seg[pos];
56         int m = (ini + fim) >> 1;
57         return f(query(e[pos], ini, m, p, q), query(d
58         [pos], m + 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 == fim) {
68             seg[pos] = val;
69             return;
70         }
71
72         int m = (ini + fim) >> 1;
73
74         if (id <= m) {
75             if (e[pos] == 0) e[pos] = create();
76             update(e[pos], ini, m, id, val);
77         } else {
78             if (d[pos] == 0) d[pos] = create();
79             update(d[pos], m + 1, fim, id, val);
80         }
81
82         seg[pos] = f(seg[e[pos]], seg[d[pos]]);
83     }
84
85     ftype query(int p, int q) {
86         return query(1, 1, n, p, q);
87     }
88
89     void update(int id, int val) {
90         update(1, 1, n, id, val);
91     }
92 };

```

### 3.4 Lazy Dynamic Implicit Sparse

```

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

```

```

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 };

```

### 3.5 Lazy

```

1 // Description:
2 // Query - get sum of elements from range (l, r)
   inclusive
3 // Update - add a value val to elementos 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 struct Segtree {
20     vector<ftype> seg;
21     vector<ftype> lazy;
22     int n;
23     const ftype NEUTRAL = 0;
24     const ftype NEUTRAL_LAZY = -1;
25
26     Segtree(int n) {
27         int sz = 1;
28         while (sz < n) sz *= 2;
29         this->n = sz;
30
31         seg.assign(2*sz, NEUTRAL);
32         lazy.assign(2*sz, NEUTRAL_LAZY);
33     }
34
35     ftype apply_lazy(ftype a, ftype b, int len) {
36         if (b == NEUTRAL_LAZY) return a;
37         if (a == NEUTRAL_LAZY) return b * len;
38         else return a + b * len;
39     }
40
41     void propagate(int pos, int ini, int fim) {
42         if (ini == fim) {
43             return;
44         }
45
46         int e = 2*pos + 1;
47         int d = 2*pos + 2;
48         int m = ini + (fim - ini) / 2;
49
50         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
51         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
52
53         seg[e] = apply_lazy(seg[e], lazy[pos], m -
54 ini + 1);
55         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
56 m);
57         lazy[pos] = NEUTRAL_LAZY;
58     }
59
60     ftype f(ftype a, ftype b) {
61         return a + b;
62     }
63
64     ftype query(int pos, int ini, int fim, int p, int
65 q) {
66         propagate(pos, ini, fim);
67
68         if (ini >= p && fim <= q) {
69             return seg[pos];
70         }
71
72         if (q < ini || p > fim) {
73             return NEUTRAL;
74         }
75
76         int e = 2*pos + 1;
77         int d = 2*pos + 2;
78         int m = ini + (fim - ini) / 2;
79
80         return f(query(e, ini, m, p, q), query(d, m +
81 1, fim, p, q));
82     }
83
84     void update(int pos, int ini, int fim, int p, int
85 q, int val) {
86         propagate(pos, ini, fim);
87
88         if (ini > q || fim < p) {
89             return;
90         }
91
92         seg[pos] = apply_lazy(seg[pos], val, fim
93 - ini + 1);
94
95         return;
96     }
97
98     int e = 2*pos + 1;
99     int d = 2*pos + 2;
100     int m = ini + (fim - ini) / 2;
101
102     update(e, ini, m, p, q, val);
103     update(d, m + 1, fim, p, q, val);
104
105     seg[pos] = f(seg[e], seg[d]);
106 }
107
108 void build(int pos, int ini, int fim, vector<int>
109 &v) {
110     if (ini == fim) {
111         if (ini < (int)v.size()) {
112             seg[pos] = v[ini];
113         }
114         return;
115     }
116
117     int e = 2*pos + 1;
118     int d = 2*pos + 2;
119     int m = ini + (fim - ini) / 2;
120
121     build(e, ini, m, v);
122     build(d, m + 1, fim, v);
123
124     seg[pos] = f(seg[e], seg[d]);
125 }
126
127 ftype query(int p, int q) {
128     return query(0, 0, n - 1, p, q);
129 }
130
131 void update(int p, int q, int val) {
132     update(0, 0, n - 1, p, q, val);
133 }
134
135 void build(vector<int> &v) {
136     build(0, 0, n - 1, v);
137 }
138
139 void debug() {
140     for (auto e : seg) {
141         cout << e << ' ';
142     }
143     cout << '\n';
144     for (auto e : lazy) {
145         cout << e << ' ';
146     }
147     cout << '\n';
148     cout << '\n';
149 }
150
151 // Description:
152 // Query - get sum of segment that is maximum among
153 // all segments
154 // E.g
155 // Array: 5 -4 4 3 -5
156 // Maximum segment sum: 8 because 5 + (-4) + 4 = 8

```

### 3.6 Segment With Maximum Sum

```

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

```

```
138 };
```

### 3.7 Segtree2d

```
1 // Description:
2 // Indexed at zero
3 // Given a N x M grid, where i represents the row and
4 // j the column, perform the following operations
5 // update(j, i) - update the value of grid[i][j]
6 // query(j1, j2, i1, i2) - return the sum of values
7 // inside the rectangle
8 // defined by grid[i1][j1] and grid[i2][j2] inclusive
9 // Problem:
10 // https://cses.fi/problemset/task/1739/
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
39         lY, int rY, vector<vector<int>> &v){
40         if(lY == rY){
41             if(lX == rX){
42                 seg[noX][noY] = v[rX][rY];
43             }else{
44                 seg[noX][noY] = seg[2*noX+1][noY] +
45                 seg[2*noX+2][noY];
46             }
47         }else{
48             int m = (lY+rY)/2;
49
50             buildY(noX, lX, rX, 2*noY+1, lY, m, v);
51             buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);
52
53             seg[noX][noY] = seg[noX][2*noY+1] + seg[
54                 noX][2*noY+2];
55         }
56     }
57
58     void buildX(int noX, int lX, int rX, vector<
59         vector<int>> &v){
60         if(lX != rX){
61             int m = (lX+rX)/2;
62
63             buildX(2*noX+1, lX, m, v);
64             buildX(2*noX+2, m+1, rX, v);
65
66             seg[noX][noY] = seg[noX][2*noY+1] + seg[
67                 noX][2*noY+2];
68         }
69     }
70
71     void updateY(int noX, int lX, int rX, int noY,
72         int lY, int rY, int y){
73         if(lY == rY){
74             if(lX == rX){
75                 seg[noX][noY] = !seg[noX][noY];
76             }else{
77                 seg[noX][noY] = seg[2*noX+1][noY] +
78                 seg[2*noX+2][noY];
79             }
80         }else{
81             int m = (lY+rY)/2;
82
83             if(y <= m){
84                 updateY(noX, lX, rX, 2*noY+1, lY, m, y
85                     );
86             }else if(m < y){
87                 updateY(noX, lX, rX, 2*noY+2, m+1, rY
88                     , y);
89             }
90
91             seg[noX][noY] = seg[noX][2*noY+1] + seg[
92                 noX][2*noY+2];
93         }
94     }
95
96     void updateX(int noX, int lX, int rX, int x, int
97         y){
98         int m = (lX+rX)/2;
99
100         if(lX != rX){
101             if(x <= m){
102                 updateX(2*noX+1, lX, m, x, y);
103             }else if(m < x){
104                 updateX(2*noX+2, m+1, rX, x, y);
105             }
106         }
107
108         updateY(noX, lX, rX, 0, 0, M - 1, y);
109     }
110
111     int queryY(int noX, int noY, int lY, int rY, int
112         aY, int bY){
113         if(aY <= lY && rY <= bY) return seg[noX][noY
114             ];
115
116         int m = (lY+rY)/2;
117
118         if(bY <= m) return queryY(noX, 2*noY+1, lY, m
119             , aY, bY);
120         if(m < aY) return queryY(noX, 2*noY+2, m+1,
121             rY, aY, bY);
122
123         return queryY(noX, 2*noY+1, lY, m, aY, bY) +
124             queryY(noX, 2*noY+2, m+1, rY, aY, bY);
125     }
126
127     int queryX(int noX, int lX, int rX, int aX, int
128         bX, int aY, int bY){
129         if(aX <= lX && rX <= bX) return queryY(noX,
130             0, 0, M - 1, aY, bY);
131
132         int m = (lX+rX)/2;
133
134         if(bX <= m) return queryX(2*noX+1, lX, m, aX,
135             bX, aY, bY);
136         if(m < aX) return queryX(2*noX+2, m+1, rX, aX
137             , bX, aY, bY);
138
139         return queryX(2*noX+1, lX, m, aX, bX, aY, bY)
140             + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
141     }
142
143     void buildY(noX, lX, rX, 0, 0, M - 1, v);
144 }
```

```
64 }
```

```
65
66 void updateY(int noX, int lX, int rX, int noY,
67 int lY, int rY, int y){
68     if(lY == rY){
69         if(lX == rX){
70             seg[noX][noY] = !seg[noX][noY];
71         }else{
72             seg[noX][noY] = seg[2*noX+1][noY] +
73             seg[2*noX+2][noY];
74         }
75     }else{
76         int m = (lY+rY)/2;
77
78         if(y <= m){
79             updateY(noX, lX, rX, 2*noY+1, lY, m, y
80                 );
81         }else if(m < y){
82             updateY(noX, lX, rX, 2*noY+2, m+1, rY
83                 , y);
84         }
85
86         seg[noX][noY] = seg[noX][2*noY+1] + seg[
87             noX][2*noY+2];
88     }
89 }
90
91 void updateX(int noX, int lX, int rX, int x, int
92 y){
93     int m = (lX+rX)/2;
94
95     if(lX != rX){
96         if(x <= m){
97             updateX(2*noX+1, lX, m, x, y);
98         }else if(m < x){
99             updateX(2*noX+2, m+1, rX, x, y);
100         }
101     }
102
103     updateY(noX, lX, rX, 0, 0, M - 1, y);
104 }
105
106 int queryY(int noX, int noY, int lY, int rY, int
107 aY, int bY){
108     if(aY <= lY && rY <= bY) return seg[noX][noY
109         ];
110
111     int m = (lY+rY)/2;
112
113     if(bY <= m) return queryY(noX, 2*noY+1, lY, m, m
114         , aY, bY);
115     if(m < aY) return queryY(noX, 2*noY+2, m+1,
116         rY, aY, bY);
117
118     return queryY(noX, 2*noY+1, lY, m, aY, bY) +
119         queryY(noX, 2*noY+2, m+1, rY, aY, bY);
120 }
121
122 int queryX(int noX, int lX, int rX, int aX, int
123 bX, int aY, int bY){
124     if(aX <= lX && rX <= bX) return queryY(noX,
125         0, 0, M - 1, aY, bY);
126
127     int m = (lX+rX)/2;
128
129     if(bX <= m) return queryX(2*noX+1, lX, m, aX,
130         bX, aY, bY);
131     if(m < aX) return queryX(2*noX+2, m+1, rX, aX
132         , bX, aY, bY);
133
134     return queryX(2*noX+1, lX, m, aX, bX, aY, bY)
135         + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
136 }
```



```

121 void build(vector<vector<int>> &v) {
122     buildX(0, 0, N - 1, v);
123 }
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 };

```

### 3.8 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
   q) {
49         if (q < ini || p > fim) return NEUTRAL;

```

```

50         if (pos == 0) return 0;
51         if (p <= ini && fim <= q) return seg[pos];
52         int m = (ini + fim) >> 1;
53         return f(query(e[pos], ini, m, p, q), query(d
   [pos], m + 1, fim, p, q));
54     }
55
56     int update(int pos, int ini, int fim, int id, int
   val) {
57         int novo = create();
58
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
   , id, val);
71         else d[novo] = update(d[novo], m + 1, fim, id
   , val);
72
73         seg[novo] = f(seg[e[novo]], seg[d[novo]]);
74
75         return novo;
76     }
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 };

```

### 3.9 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 }

```

### 3.10 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>;

```

### 3.11 Priority Queue

```

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

```

## 4 Template

### 4.1 Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define int long long
5 #define optimize std::ios::sync_with_stdio(false);
   cin.tie(NULL);
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 }

```

### 4.2 Template Clean

```

1 // Notes:
2 // Compile and execute
3 // g++ teste.cpp -o teste -std=c++17
4 // ./teste < teste.txt
5
6 // Print with precision
7 // cout << fixed << setprecision(12) << value << endl
8 ;
9
10 // File as input and output
11 // freopen("input.txt", "r", stdin);
12 // freopen("output.txt", "w", stdout);
13
14 #include <bits/stdc++.h>
15 using namespace std;
16
17 int main() {
18     ios::sync_with_stdio(false);
19     cin.tie(NULL);
20
21
22     return 0;
23 }

```

## 5 Graphs

### 5.1 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 = 0x3f3f3f3f3f3f3f3fLL;
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 }

```

### 5.2 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 }

```

### 5.3 Cycle Path Recovery

```

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

```

```

45         for (int v : cycle)
46             cout << v << " ";
47         cout << endl;
48     }
49 }

```

## 5.4 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 }

```

## 5.5 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
35     return false;
36 }

```

## 5.6 Dinic

```

1  const int N = 300;
2
3  struct Dinic {
4      struct Edge{
5          int from, to; ll flow, cap;
6      };
7      vector<Edge> edge;
8
9      vector<int> g[N];
10     int ne = 0;
11     int lvl[N], vis[N], pass;
12     int qu[N], px[N], qt;
13
14     ll run(int s, int sink, ll minE) {
15         if(s == sink) return minE;
16
17         ll ans = 0;
18
19         for(; px[s] < (int)g[s].size(); px[s]++) {
20             int e = g[s][ px[s] ];
21             auto &v = edge[e], &rev = edge[e^1];
22             if(lvl[v.to] != lvl[s]+1 || v.flow >= v.
23                 cap)
24                 continue; // v.cap - v.flow
25                 < lim
26                 ll tmp = run(v.to, sink,min(minE, v.cap-v
27                     .flow));
28                 v.flow += tmp, rev.flow -= tmp;
29                 ans += tmp, minE -= tmp;
30                 if(minE == 0) break;
31             }
32             return ans;
33         }
34     }
35     bool bfs(int source, int sink) {
36         qt = 0;
37         qu[qt++] = source;
38         lvl[source] = 1;
39         vis[source] = ++pass;
40         for(int i = 0; i < qt; i++) {
41             int u = qu[i];
42             px[u] = 0;
43             if(u == sink) return true;
44             for(auto& ed : g[u]) {
45                 auto v = edge[ed];
46                 if(v.flow >= v.cap || vis[v.to] ==
47                     pass)
48                     continue; // v.cap - v.flow < lim
49                 vis[v.to] = pass;
50                 lvl[v.to] = lvl[u]+1;
51                 qu[qt++] = v.to;
52             }
53             return false;
54         }
55     }
56     ll flow(int source, int sink) {
57         reset_flow();
58         ll ans = 0;
59         //for(lim = (1LL << 62); lim >= 1; lim /= 2)
60         while(bfs(source, sink))

```

```

56         ans += run(source, sink, LLINF);
57         return ans;
58     }
59     void addEdge(int u, int v, ll c, ll rc) {
60         Edge e = {u, v, 0, c};
61         edge.pb(e);
62         g[u].push_back(ne++);
63
64         e = {v, u, 0, rc};
65         edge.pb(e);
66         g[v].push_back(ne++);
67     }
68     void reset_flow() {
69         for(int i = 0; i < ne; i++)
70             edge[i].flow = 0;
71         memset(lvl, 0, sizeof(lvl));
72         memset(vis, 0, sizeof(vis));
73         memset(qu, 0, sizeof(qu));
74         memset(px, 0, sizeof(px));
75         qt = 0; pass = 0;
76     }
77 };

```

## 5.7 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 }

```

## 5.8 Dijkstra

```

1 const int MAX = 2e5+7;
2 const int INF = 1000000000;
3 vector<vector<pair<int, int>>> adj(MAX);
4
5 void dijkstra(int s, vector<int> & d, vector<int> & p
6 ) {
7     int n = adj.size();
8     d.assign(n, INF);
9     p.assign(n, -1);
10
11     d[s] = 0;
12     set<pair<int, int>> q;
13     q.insert({0, s});
14     while (!q.empty()) {
15         int v = q.begin()->second;
16         q.erase(q.begin());
17
18         for (auto edge : adj[v]) {
19             int to = edge.first;
20             int len = edge.second;
21
22             if (d[v] + len < d[to]) {
23                 q.erase({d[to], to});
24                 d[to] = d[v] + len;
25                 p[to] = v;

```

```

26                 q.insert({d[to], to});
27             }
28         }
29     }
30
31     vector<int> restore_path(int s, int t) {
32         vector<int> path;
33
34         for (int v = t; v != s; v = p[v])
35             path.push_back(v);
36         path.push_back(s);
37
38         reverse(path.begin(), path.end());
39         return path;
40     }
41
42     int adj[MAX][MAX];
43     int dist[MAX];
44     int minDistance(int dist[], bool sptSet[], int V) {
45         int min = INT_MAX, min_index;
46
47         for (int v = 0; v < V; v++)
48             if (sptSet[v] == false && dist[v] <= min)
49                 min = dist[v], min_index = v;
50
51         return min_index;
52     }
53
54     void dijkstra(int src, int V) {
55         bool sptSet[V];
56         for (int i = 0; i < V; i++)
57             dist[i] = INT_MAX, sptSet[i] = false;
58
59         dist[src] = 0;
60
61         for (int count = 0; count < V - 1; count++) {
62             int u = minDistance(dist, sptSet, V);
63
64             sptSet[u] = true;
65
66             for (int v = 0; v < V; v++)
67                 if (!sptSet[v] && adj[u][v]
68                     && dist[u] != INT_MAX
69                     && dist[u] + adj[u][v] < dist[v])
70                     dist[v] = dist[u] + adj[u][v];
71         }
72     }
73 }
74 }

```

## 5.9 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) {

```

```

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 }

```

## 5.10 Centroid Find

```

1 // Description:
2 // Indexed at zero
3 // Find a centroid, that is a node such that when it
4 // is appointed the root of the tree,
5 // each subtree has at most floor(n/2) nodes.
6 // Problem:
7 // https://cses.fi/problemset/task/2079/
8 // Complexity:
9 // O(n)
10 // How to use:
11 // get_subtree_size(0);
12 // cout << get_centroid(0) + 1 << endl;
13
14 int n;
15 vector<int> adj[MAX];
16 int subtree_size[MAX];
17
18 int get_subtree_size(int node, int par = -1) {
19     int &res = subtree_size[node];
20     res = 1;
21     for (int i : adj[node]) {
22         if (i == par) continue;
23         res += get_subtree_size(i, node);
24     }
25     return res;
26 }
27
28 int get_centroid(int node, int par = -1) {
29     for (int i : adj[node]) {
30         if (i == par) continue;
31         if (subtree_size[i] * 2 > n) { return
32             get_centroid(i, node); }
33     }
34     return node;
35 }
36
37 int main() {
38     cin >> n;
39     for (int i = 0; i < n - 1; i++) {
40         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 }

```

## 5.11 Small To Large

```

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

## 5.12 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].
w < min_e[v].w))
19                 v = j;
20         }
21
22         if (min_e[v].w == INF) {
23             cout << "No MST!" << endl;
24             exit(0);
25         }
26
27         selected[v] = true;
28         total_weight += min_e[v].w;
29         if (min_e[v].to != -1)
30             cout << v << " " << min_e[v].to << endl;
31
32         for (int to = 0; to < n; ++to) {
33             if (adj[v][to] < min_e[to].w)
34                 min_e[to] = {adj[v][to], v};
35         }
36     }
37
38     cout << total_weight << endl;
39 }
```

## 5.13 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<(Edge const& 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 }
```

## 5.14 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 //  $\text{dist}[a] + \text{dist}[b] - 2 \cdot \text{dist}[\text{lca}(a, b)]$ 
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 }

```

## 5.15 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 }

```

## 6 Strings

### 6.1 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 }

```



## 6.2 Lcs

```
1 // Description:
2 // Finds the longest common subsequence between two
  string
3
4 // Problem:
5 // https://codeforces.com/gym/103134/problem/B
6
7 // Complexity:
8 // O(mn) where m and n are the length of the strings
9
10 string lcsAlgo(string s1, string s2, int m, int n) {
11     int LCS_table[m + 1][n + 1];
12
13     for (int i = 0; i <= m; i++) {
14         for (int j = 0; j <= n; j++) {
15             if (i == 0 || j == 0)
16                 LCS_table[i][j] = 0;
17             else if (s1[i - 1] == s2[j - 1])
18                 LCS_table[i][j] = LCS_table[i - 1][j - 1] +
19                 1;
20             else
21                 LCS_table[i][j] = max(LCS_table[i - 1][j],
22                                     LCS_table[i][j - 1]);
23         }
24     }
25
26     int index = LCS_table[m][n];
27     char lcsAlgo[index + 1];
28     lcsAlgo[index] = '\0';
29
30     int i = m, j = n;
31     while (i > 0 && j > 0) {
32         if (s1[i - 1] == s2[j - 1]) {
33             lcsAlgo[index - 1] = s1[i - 1];
34             i--;
35             j--;
36             index--;
37         }
38         else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
39             i--;
40         else
41             j--;
42     }
43
44     return lcsAlgo;
45 }
```

## 6.3 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 }
```

## 6.4 Generate All Sequences Length K

```
1 // gera todas as ípossveis êsequencias usando as letras
2 // em set (de comprimento n) e que tenham tamanho k
3 // 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
16     return ans;
17 }
```

## 6.5 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 }
```

## 7 DP

### 7.1 Knapsack

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

### 7.2 Substr Palindrome

```
1 // êvoc deve informar se a substring de S formada
2 // pelos elementos entre os indices i e j
3 // ê um palindromo ou ãno.
4
5 char s[MAX];
6 int calculado[MAX][MAX]; // inciado com false, ou 0
7 int tabela[MAX][MAX];
8
9 int is_palin(int i, int j){
10     if(calculado[i][j]){
11         return tabela[i][j];
12     }
13     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 }

```

## 7.3 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 }

```

## 7.4 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

```

```

13                 K[i][w] = K[i - 1][w];
14             }
15         }
16     }
17     int res = K[n][W];
18     cout<< res << endl;
19
20     w = W;
21     for (i = n; i > 0 && res > 0; i--) {
22         if (res == K[i - 1][w])
23             continue;
24         else {
25             cout<<" "<<wt[i - 1] ;
26             res = res - val[i - 1];
27             w = w - wt[i - 1];
28         }
29     }
30 }
31
32 int main()
33 {
34     int val[] = { 60, 100, 120 };
35     int wt[] = { 10, 20, 30 };
36     int W = 50;
37     int n = sizeof(val) / sizeof(val[0]);
38
39     knapsack(W, wt, val, n);
40
41     return 0;
42 }

```

## 7.5 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    }
18    return tabela[k] = melhor;

```

## 7.6 Digits

```

1 // achar a quantidade de numeros menores que R que
  possuem no maximo 3 digitos nao nulos
2 // a ideia eh utilizar da ordem lexicografica para
  checar isso pois se temos por exemplo
3 // o numero 8500, a gente sabe que se pegarmos o
  numero 7... qualquer digito depois do 7
4 // sera necessariamente menor q 8500
5
6 string r;
7 int tab[20][2][5];
8
9 // i - digito de R
10 // menor - ja pegou um numero menor que um digito de
  R
11 // qt - quantidade de digitos nao nulos
12 int dp(int i, bool menor, int qt){
13     if(qt > 3) return 0;

```

```

14     if(i >= r.size()) return 1;
15     if(tab[i][menor][qt] != -1) return tab[i][menor][
qt];
16
17     int dr = r[i]-'0';
18     int res = 0;
19
20     for(int d = 0; d <= 9; d++) {
21         int dnn = qt + (d > 0);
22         if(menor == true) {
23             res += dp(i+1, true, dnn);
24         }
25         else if(d < dr) {
26             res += dp(i+1, true, dnn);
27         }
28         else if(d == dr) {
29             res += dp(i+1, false, dnn);
30         }
31     }
32
33     return tab[i][menor][qt] = res;
34 }

```

## 7.7 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 }

```

## 7.8 Kadane

```

1 // achar uma subsequencia continua no array que a
soma seja a maior possivel
2 // nesse caso vc precisa multiplicar exatamente 1
elemento da subsequencia
3 // e achar a maior soma com isso
4
5 int n, x, arr[MAX], tab[MAX][2]; // tab[maior
resposta no intervalo][foi multiplicado ou ão]
6
7 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 }

```

## 8 Math

### 8.1 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
entao phi(m) = p-1
23     ll e = phim-1;
24     return fexp(a, e, MOD);
25 }

```

### 8.2 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 }

```

## 8.3 Prime Factors

```

1 vector<pair<long long, int>> fatora(long long n) {
2     vector<pair<long long, int>> ans;
3     for(long long p = 2; p*p <= n; p++) {
4         if(n % p == 0) {
5             int expoente = 0;
6             while(n % p == 0) {
7                 n /= p;
8                 expoente++;
9             }
10            ans.emplace_back(p, expoente);
11        }
12    }
13    if(n > 1) ans.emplace_back(n, 1);
14    return ans;
15 }

```

## 8.4 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 }

```

## 8.5 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 }

```

## 8.6 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 }

```

## 8.7 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
8         solution
9         ans = ans + x * ((a - ans) / g) % (b / g) *
10        lcm;
11        lcm = lcm * (b / g);
12        ans = (ans % lcm + lcm) % lcm;
13    }
14    return ans;
15 }

```

## 8.8 Ceil

```

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

```

## 8.9 Matrix Exponentiation

```

1 // Description:
2 // Calculate the nth term of a linear recursion
3
4 // Example Fibonacci:
5 // Given a linear recurrence, for example fibonacci
6 // F(n) = n, x <= 1
7 // F(n) = F(n - 1) + F(n - 2), x > 1
8
9 // The recurrence has two terms, so we can build a
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 }

```

## 8.10 Linear Diophantine Equation

```

1 // int a, b, c, x1, x2, y1, y2; cin >> a >> b >> c >>
    x1 >> x2 >> y1 >> y2;
2 // int ans = -1;
3 // 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
    passed as reference
40 // minx <= x <= maxx miny <= y <= maxy
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 }

```

## 8.11 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 }

```

# 9 Algorithms

## 9.1 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 }

```

## 9.2 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         double f2 = f(m2); //evaluates the
9         function at m1
10        function at m2
11        if (f1 < f2)
12            l = m1;
13        else
14            r = m2;
15    }
16    return f(l); //return the
17    maximum of f(x) in [l, r]
18 }

```

## 9.3 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 }

```

## 9.4 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 }

```

## 9.5 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;

```