



Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1 Misc	2	4.9 Dijkstra	16
1.1 Split	2	4.10 Tarjan Bridge	17
1.2 Int128	2	4.11 Centroid Find	17
2 Data Structures	2	4.12 Small To Large	17
2.1 Range Query Point Update	2	4.13 2sat	18
2.2 Lazy Assignment To Segment	3	4.14 Prim	19
2.3 Minimum And Amount	4	4.15 Kruskall	20
2.4 Dynamic Implicit Sparse	5	4.16 Lca	20
2.5 Lazy Dynamic Implicit Sparse	5	4.17 Centroid Decomposition	21
2.6 Segment With Maximum Sum	6	5 Strings	22
2.7 Segtree2d	7	5.1 Kmp	22
2.8 Lazy Addition To Segment	8	5.2 Lcs	22
2.9 Persistent	10	5.3 Generate All Permutations	22
2.10 Dsu	10	5.4 Generate All Sequences Length K	22
2.11 Ordered Set	11	5.5 Trie	22
2.12 Two Sets	11	5.6 Z-function	23
2.13 Priority Queue	12	6 DP	23
3 Template	12	6.1 Knapsack	23
3.1 Template	12	6.2 Substr Palindrome	23
3.2 Template Clean	12	6.3 Edit Distance	23
4 Graphs	12	6.4 Knapsack With Index	24
4.1 Floyd Warshall	12	6.5 Minimum Coin Change	24
4.2 Tree Diameter	13	6.6 Digits	24
4.3 Cycle Path Recovery	13	6.7 Coins	24
4.4 Ford Fulkerson Edmonds Karp	13	6.8 Kadane	24
4.5 Bipartite	14	7 Math	25
4.6 Find Cycle	14	7.1 Function Root	25
4.7 Dinic	14	7.2 Multiplicative Inverse	25
4.8 Bellman Ford	16	7.3 Divisors	25
		7.4 Prime Factors	25

7.5	Binary To Decimal	26
7.6	Horner Algorithm	26
7.7	Sieve Of Eratosthenes	26
7.8	Check If Bit Is On	26
7.9	Crt	26
7.10	Ceil	26
7.11	Matrix Exponentiation	26
7.12	Linear Diophantine Equation	27
7.13	Representation Arbitrary Base	28
7.14	To Decimal	28
7.15	Fast Exponentiation	28
8	Algorithms	28
8.1	Lis	28
8.2	Ternary Search	28
8.3	Binary Search First True	29
8.4	Delta-encoding	29
8.5	Binary Search Last True	29
8.6	Biggest K	29

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 Data Structures

2.1 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:
11 // O(log n) for both query and update
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(l, 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         seg[pos] = f(seg[e], seg[d]);
83     }
84 }
```

```

80 void build(int pos, int ini, int fim, vector<int>
    &v) {
81     if (ini == fim) {
82         if (ini < (int)v.size()) {
83             seg[pos] = v[ini];
84         }
85         return;
86     }
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 };

```

2.2 Lazy Assignment To Segment

```

1  const long long INF = 1e18+10;
2
3  typedef long long ftype;
4
5  struct Segtree {
6      vector<ftype> seg;
7      vector<ftype> lazy;
8      int n;
9      const ftype NEUTRAL = 0;
10     const ftype NEUTRAL_LAZY = -INF;
11
12     Segtree(int n) {
13         int sz = 1;
14         // potencia de dois mais proxima
15         while (sz < n) sz *= 2;
16         this->n = sz;
17
18         // numero de nos da seg
19         seg.assign(2*sz, NEUTRAL);
20         lazy.assign(2*sz, NEUTRAL_LAZY);
21     }
22
23     ftype apply_lazy(ftype a, ftype b, int len) {
24         if (b == NEUTRAL_LAZY) return a;
25         if (a == NEUTRAL_LAZY) return b * len;
26         else return b * len;
27     }
28
29     void propagate(int pos, int ini, int fim) {
30         if (ini == fim) {
31             return;
32         }

```

```

33         int e = 2*pos + 1;
34         int d = 2*pos + 2;
35         int m = ini + (fim - ini) / 2;
36
37         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
38         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
39
40         seg[e] = apply_lazy(seg[e], lazy[pos], m -
41             ini + 1);
42         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
43             m);
44
45         lazy[pos] = NEUTRAL_LAZY;
46     }
47
48     ftype f(ftype a, ftype b) {
49         return a + b;
50     }
51
52     ftype query(int pos, int ini, int fim, int p, int
53         q) {
54         propagate(pos, ini, fim);
55
56         if (ini >= p && fim <= q) {
57             return seg[pos];
58         }
59
60         if (q < ini || p > fim) {
61             return NEUTRAL;
62         }
63
64         int e = 2*pos + 1;
65         int d = 2*pos + 2;
66         int m = ini + (fim - ini) / 2;
67
68         return f(query(e, ini, m, p, q), query(d, m +
69             1, fim, p, q));
70     }
71
72     void update(int pos, int ini, int fim, int p, int
73         q, int val) {
74         propagate(pos, ini, fim);
75
76         if (ini > q || fim < p) {
77             return;
78         }
79
80         if (ini >= p && fim <= q) {
81             lazy[pos] = apply_lazy(lazy[pos], val, 1)
82             ;
83             seg[pos] = apply_lazy(seg[pos], val, fim
84                 - ini + 1);
85             return;
86         }
87
88         int e = 2*pos + 1;
89         int d = 2*pos + 2;
90         int m = ini + (fim - ini) / 2;
91
92         update(e, ini, m, p, q, val);
93         update(d, m + 1, fim, p, q, val);
94
95         seg[pos] = f(seg[e], seg[d]);
96     }
97
98     void build(int pos, int ini, int fim, vector<int>
99         &v) {
100         if (ini == fim) {
101             // se a posição existir no array original
102             // seg tamanho potencia de dois
103             if (ini < (int)v.size()) {

```

```

98         seg[pos] = v[ini];
99     }
100     return;
101 }
102
103 int e = 2*pos + 1;
104 int d = 2*pos + 2;
105 int m = ini + (fim - ini) / 2;
106
107 build(e, ini, m, v);
108 build(d, m + 1, fim, v);
109
110 seg[pos] = f(seg[e], seg[d]);
111 }
112
113 ftype query(int p, int q) {
114     return query(0, 0, n - 1, p, q);
115 }
116
117 void update(int p, int q, int val) {
118     update(0, 0, n - 1, p, q, val);
119 }
120
121 void build(vector<int> &v) {
122     build(0, 0, n - 1, v);
123 }
124
125 void debug() {
126     for (auto e : seg) {
127         cout << e << ' ';
128     }
129     cout << '\n';
130     for (auto e : lazy) {
131         cout << e << ' ';
132     }
133     cout << '\n';
134     cout << '\n';
135 }
136 };

```

2.3 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
   q) {
46         if (ini >= p && fim <= q) {
47             return seg[pos];
48         }
49
50         if (q < ini || p > fim) {
51             return NEUTRAL;
52         }
53
54         int e = 2*pos + 1;
55         int d = 2*pos + 2;
56         int m = ini + (fim - ini) / 2;
57
58         return f(query(e, ini, m, p, q), query(d, m +
   1, fim, p, q));
59     }
60
61     void update(int pos, int ini, int fim, int id,
   int val) {
62         if (ini > id || fim < id) {
63             return;
64         }
65
66         if (ini == id && fim == id) {
67             seg[pos] = mp(val, 1);
68
69             return;
70         }
71
72         int e = 2*pos + 1;
73         int d = 2*pos + 2;
74         int m = ini + (fim - ini) / 2;
75
76         update(e, ini, m, id, val);
77         update(d, m + 1, fim, id, val);
78
79         seg[pos] = f(seg[e], seg[d]);
80     }
81
82     void build(int pos, int ini, int fim, vector<int>
   &v) {
83         if (ini == fim) {
84             if (ini < (int)v.size()) {
85                 seg[pos] = mp(v[ini], 1);
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         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 };

```

2.4 Dynamic Implicit Sparse

```

1 // Description:
2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
5 // stored in an array
6 // and the queries need to be answered online so we
7 // can't sort the nodes and compress them
8 // we create nodes only when they are needed so there
9 // 'll be (Q*log(MAX)) nodes
10 // where Q is the number of queries and MAX is the
11 // maximum index a node can assume
12
13 // Query - get sum of elements from range (l, r)
14 // inclusive
15 // Update - update element at position id to a value
16 // val
17
18 // Problem:
19 // https://cses.fi/problemset/task/1648
20
21 // Complexity:
22 // O(log n) for both query and update
23
24 // How to use:
25 // MAX is the maximum index a node can assume
26
27 // Segtree seg = Segtree(MAX);
28
29 typedef long long ftype;
30
31 const int MAX = 1e9+17;
32
33 struct Segtree {
34     vector<ftype> seg, d, e;
35     const ftype NEUTRAL = 0;
36     int n;
37
38     Segtree(int n) {
39         this->n = n;
40         create();
41         create();
42     }
43
44     ftype f(ftype a, ftype b) {
45         return a + b;
46     }
47
48     ftype create() {

```

```

43     seg.push_back(0);
44     e.push_back(0);
45     d.push_back(0);
46     return seg.size() - 1;
47 }
48
49 ftype query(int pos, int ini, int fim, int p, int
50 q) {
51     if (q < ini || p > fim) return NEUTRAL;
52     if (pos == 0) return 0;
53     if (p <= ini && fim <= q) return seg[pos];
54     int m = (ini + fim) >> 1;
55     return f(query(e[pos], ini, m, p, q), query(d
56 [pos], m + 1, fim, p, q));
57 }
58
59 void update(int pos, int ini, int fim, int id,
60 int val) {
61     if (ini > id || fim < id) {
62         return;
63     }
64
65     if (ini == fim) {
66         seg[pos] = val;
67     }
68
69     return;
70
71     int m = (ini + fim) >> 1;
72
73     if (id <= m) {
74         if (e[pos] == 0) e[pos] = create();
75         update(e[pos], ini, m, id, val);
76     } else {
77         if (d[pos] == 0) d[pos] = create();
78         update(d[pos], m + 1, fim, id, val);
79     }
80
81     seg[pos] = f(seg[e[pos]], seg[d[pos]]);
82 }
83
84 ftype query(int p, int q) {
85     return query(1, 1, n, p, q);
86 }
87
88 void update(int id, int val) {
89     update(1, 1, n, id, val);
90 }
91
92 };

```

2.5 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
17
18 // Problem:
19 // https://oj.uz/problem/view/IZh012_apple
20
21 // 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 const int MAX = 1e9+10;
28 const long long INF = 1e18+10;
29
30 typedef long long ftype;
31
32 struct Segtree {
33     vector<ftype> seg, d, e, lazy;
34     const ftype NEUTRAL = 0;
35     const ftype NEUTRAL_LAZY = -INF;
36     int n;
37
38     Segtree(int n) {
39         this->n = n;
40     }
41
42     ftype apply_lazy(ftype a, ftype b, int len) {
43         if (b == NEUTRAL_LAZY) return a;
44         else return b * len;
45     }
46
47     void propagate(int pos, int ini, int fim) {
48         if (seg[pos] == 0) return;
49
50         if (ini == fim) {
51             return;
52         }
53
54         int m = (ini + fim) >> 1;
55
56         if (e[pos] == 0) e[pos] = create();
57         if (d[pos] == 0) d[pos] = create();
58
59         lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[
pos], 1);
60         lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[
pos], 1);
61
62         seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[
pos], m - ini + 1);
63         seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[
pos], fim - m);
64
65         lazy[pos] = NEUTRAL_LAZY;
66     }
67
68     ftype f(ftype a, ftype b) {
69         return a + b;
70     }
71
72     ftype create() {
73         seg.push_back(0);
74         e.push_back(0);
75         d.push_back(0);
76         lazy.push_back(-1);
77         return seg.size() - 1;
78     }
79
80     ftype query(int pos, int ini, int fim, int p, int
q) {
81         propagate(pos, ini, fim);
82         if (q < ini || p > fim) return NEUTRAL;
83         if (pos == 0) return 0;

```

```

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

```

2.6 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(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,
35         ftype_node suf, ftype_node sum) : max_seg(max_seg),
36         pref(pref), suf(suf), sum(sum) {};
37 };
38
39 typedef Node ftype;
40
41 struct Segtree {
42     vector<ftype> seg;
43     int n;
44     const ftype NEUTRAL = Node(0, 0, 0, 0);
45
46     Segtree(int n) {
47         int sz = 1;
48         // potencia de dois mais proxima
49         while (sz < n) sz *= 2;
50         this->n = sz;
51
52         // numero de nos da seg
53         seg.assign(2*sz, NEUTRAL);
54     }
55
56     ftype f(ftype a, ftype b) {
57         ftype_node max_seg = max({a.max_seg, b.
58 max_seg, a.suf + b.pref});
59         ftype_node pref = max(a.pref, a.sum + b.pref);
60
61         ftype_node suf = max(b.suf, b.sum + a.suf);
62         ftype_node sum = a.sum + b.sum;
63
64         return Node(max_seg, pref, suf, sum);
65     }
66
67     ftype query(int pos, int ini, int fim, int p, int
68 q) {
69     if (ini >= p && fim <= q) {
70         return seg[pos];
71     }
72
73     if (q < ini || p > fim) {
74         return NEUTRAL;
75     }
76
77     int e = 2*pos + 1;
78     int d = 2*pos + 2;
79     int m = ini + (fim - ini) / 2;
80
81     return f(query(e, ini, m, p, q), query(d, m +
82 1, fim, p, q));
83 }
84
85 void update(int pos, int ini, int fim, int id,
86 int val) {
87     if (ini > id || fim < id) {
88         return;
89     }
90
91     if (ini == id && fim == id) {
92         seg[pos] = Node(val, val, val, val);
93
94         return;
95     }
96
97     int e = 2*pos + 1;
98
99
100
101     int d = 2*pos + 2;
102     int m = ini + (fim - ini) / 2;
103
104     update(e, ini, m, id, val);
105     update(d, m + 1, fim, id, val);
106
107     seg[pos] = f(seg[e], seg[d]);
108 }
109
110 void build(int pos, int ini, int fim, vector<int>
111 &v) {
112     if (ini == fim) {
113         // se a posição existir no array original
114         // seg tamanho potencia de dois
115         if (ini < (int)v.size()) {
116             seg[pos] = Node(v[ini], v[ini], v[ini],
117 v[ini]);
118         }
119         return;
120     }
121
122     int e = 2*pos + 1;
123     int d = 2*pos + 2;
124     int m = ini + (fim - ini) / 2;
125
126     build(e, ini, m, v);
127     build(d, m + 1, fim, v);
128
129     seg[pos] = f(seg[e], seg[d]);
130 }
131
132 ftype query(int p, int q) {
133     return query(0, 0, n - 1, p, q);
134 }
135
136 void update(int id, int val) {
137     update(0, 0, n - 1, id, val);
138 }
139
140 void build(vector<int> &v) {
141     build(0, 0, n - 1, v);
142 }
143
144 void debug() {
145     for (auto e : seg) {
146         cout << e.max_seg << ' ' << e.pref << ' '
147 << e.suf << ' ' << e.sum << '\n';
148     }
149     cout << '\n';
150 }
151
152 };

```

2.7 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
10 // Problem:
11 // https://cses.fi/problemset/task/1739/
12
13 // Complexity:
14 // Time complexity:
15 // O(log N * log M) for both query and update
16 // O(N * M) for build
17 // Memory complexity:
18 // 4 * M * N
19

```



```

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             seg[noX][noY] = seg[noX][2*noY+1] + seg[
53             noX][2*noY+2];
54         }
55     }
56
57     void buildX(int noX, int lX, int rX, vector<
58     vector<int>> &v){
59         if(lX != rX){
60             int m = (lX+rX)/2;
61
62             buildX(2*noX+1, lX, m, v);
63             buildX(2*noX+2, m+1, rX, v);
64         }
65
66         buildY(noX, lX, rX, 0, 0, M - 1, v);
67     }
68
69     void updateY(int noX, int lX, int rX, int noY,
70     int lY, int rY, int y){
71         if(lY == rY){
72             if(lX == rX){
73                 seg[noX][noY] = !seg[noX][noY];
74             }else{
75                 seg[noX][noY] = seg[2*noX+1][noY] +
76                 seg[2*noX+2][noY];
77             }
78         }else{
79             int m = (lY+rY)/2;
80
81             if(y <= m){
82                 updateY(noX, lX, rX, 2*noY+1, lY, m, y
83             );
84             }else if(m < y){
85                 updateY(noX, lX, rX, 2*noY+2, m+1, rY
86             , y);
87             }
88
89             seg[noX][noY] = seg[noX][2*noY+1] + seg[

```

```

83         noX][2*noY+2];
84     }
85
86     void updateX(int noX, int lX, int rX, int x, int
87     y){
88         int m = (lX+rX)/2;
89
90         if(lX != rX){
91             if(x <= m){
92                 updateX(2*noX+1, lX, m, x, y);
93             }else if(m < x){
94                 updateX(2*noX+2, m+1, rX, x, y);
95             }
96         }
97
98         updateY(noX, lX, rX, 0, 0, M - 1, y);
99     }
100
101     int queryY(int noX, int noY, int lY, int rY, int
102     aY, int bY){
103         if(aY <= lY && rY <= bY) return seg[noX][noY
104         ];
105
106         int m = (lY+rY)/2;
107
108         if(bY <= m) return queryY(noX, 2*noY+1, lY, m
109         , aY, bY);
110         if(m < aY) return queryY(noX, 2*noY+2, m+1,
111         rY, aY, bY);
112
113         return queryY(noX, 2*noY+1, lY, m, aY, bY) +
114         queryY(noX, 2*noY+2, m+1, rY, aY, bY);
115     }
116
117     int queryX(int noX, int lX, int rX, int aX, int
118     bX, int aY, int bY){
119         if(aX <= lX && rX <= bX) return queryY(noX,
120         0, 0, M - 1, aY, bY);
121
122         int m = (lX+rX)/2;
123
124         if(bX <= m) return queryX(2*noX+1, lX, m, aX,
125         bX, aY, bY);
126         if(m < aX) return queryX(2*noX+2, m+1, rX, aX
127         , bX, aY, bY);
128
129         return queryX(2*noX+1, lX, m, aX, bX, aY, bY)
130         + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
131     }
132
133     void build(vector<vector<int>> &v) {
134         buildX(0, 0, N - 1, v);
135     }
136
137     int query(int aX, int bX, int aY, int bY) {
138         return queryX(0, 0, N - 1, aX, bX, aY, bY);
139     }
140
141     void update(int x, int y) {
142         updateX(0, 0, N - 1, x, y);
143     }
144 };

```

2.8 Lazy Addition To Segment

```

1 // Description:
2 // Query - get sum of elements from range (l, r)
3 // inclusive
4 // Update - add a value val to elementos from range (
5 // l, r) inclusive
6 // 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 const long long INF = 1e18+10;
19
20 typedef long long ftype;
21
22 struct Segtree {
23     vector<ftype> seg;
24     vector<ftype> lazy;
25     int n;
26     const ftype NEUTRAL = 0;
27     const ftype NEUTRAL_LAZY = -INF;
28
29     Segtree(int n) {
30         int sz = 1;
31         while (sz < n) sz *= 2;
32         this->n = sz;
33
34         seg.assign(2*sz, NEUTRAL);
35         lazy.assign(2*sz, NEUTRAL_LAZY);
36     }
37
38     ftype apply_lazy(ftype a, ftype b, int len) {
39         if (b == NEUTRAL_LAZY) return a;
40         if (a == NEUTRAL_LAZY) return b * len;
41         else return a + b * len;
42     }
43
44     void propagate(int pos, int ini, int fim) {
45         if (ini == fim) {
46             return;
47         }
48
49         int e = 2*pos + 1;
50         int d = 2*pos + 2;
51         int m = ini + (fim - ini) / 2;
52
53         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
54         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
55
56         seg[e] = apply_lazy(seg[e], lazy[pos], m -
ini + 1);
57         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
m);
58
59         lazy[pos] = NEUTRAL_LAZY;
60     }
61
62     ftype f(ftype a, ftype b) {
63         return a + b;
64     }
65
66     ftype query(int pos, int ini, int fim, int p, int
q) {
67         propagate(pos, ini, fim);
68
69         if (ini >= p && fim <= q) {
70             return seg[pos];
71         }
72
73         if (q < ini || p > fim) {
74             return NEUTRAL;
75         }
76
77         int e = 2*pos + 1;
78         int d = 2*pos + 2;
79         int m = ini + (fim - ini) / 2;
80
81         return f(query(e, ini, m, p, q), query(d, m +
1, fim, p, q));
82     }
83
84     void update(int pos, int ini, int fim, int p, int
q, int val) {
85         propagate(pos, ini, fim);
86
87         if (ini > q || fim < p) {
88             return;
89         }
90
91         if (ini >= p && fim <= q) {
92             lazy[pos] = apply_lazy(lazy[pos], val, 1)
;
93             seg[pos] = apply_lazy(seg[pos], val, fim
- 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>
&v) {
109         if (ini == fim) {
110             if (ini < (int)v.size()) {
111                 seg[pos] = v[ini];
112             }
113             return;
114         }
115
116         int e = 2*pos + 1;
117         int d = 2*pos + 2;
118         int m = ini + (fim - ini) / 2;
119
120         build(e, ini, m, v);
121         build(d, m + 1, fim, v);
122
123         seg[pos] = f(seg[e], seg[d]);
124     }
125
126     ftype query(int p, int q) {
127         return query(0, 0, n - 1, p, q);
128     }
129
130     void update(int p, int q, int val) {
131         update(0, 0, n - 1, p, q, val);
132     }
133
134     void build(vector<int> &v) {
135         build(0, 0, n - 1, v);
136     }
137
138     void debug() {
139         for (auto e : seg) {
140             cout << e << ' ';
141         }

```

```

142         cout << '\n';
143         for (auto e : lazy) {
144             cout << e << ' ';
145         }
146         cout << '\n';
147         cout << '\n';
148     }
149 };

```

2.9 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 };

```

2.10 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         if (sizes[a] < sizes[b])
37             swap(a, b);
38
39         sizes[a] += sizes[b];
40         link[b] = a;
41     }
42
43     int size(int x) {
44         return sizes[x];
45     }
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

```

2.11 Ordered Set

```

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

```

```

26 #include <ext/pb_ds/tree_policy.hpp>
27
28 using namespace __gnu_pbds;
29 template <typename T>
30 using ordered_set = tree<T, null_type, less<T>,
   rb_tree_tag, tree_order_statistics_node_update>;
31
32 void Erase(ordered_set<int>& a, int x){
33     int r = a.order_of_key(x);
34     auto it = a.find_by_order(r);
35     a.erase(it);
36 }

```

2.12 Two Sets

```

1 // Description
2 // The values are divided in two multisets so that
   one of them contain all values that are
3 // smaller than the median and the other one contains
   all values that are greater or equal to the
   median.
4
5 // Problem:
6 // https://atcoder.jp/contests/abc306/tasks/abc306_e
7 // Problem I - Maratona Feminina de çãProgramao da
   Unicamp 2023
8 // https://codeforces.com/group/WYIydkIPyE/contest
   /450037/attachments
9
10 // Complexity:
11 // Add and remove elements - O(log n)
12 // Return sum of biggest or smallest set or return
   the median - O(1)
13
14 using ll = long long;
15
16 struct TwoSets {
17     multiset<int> small;
18     multiset<int> big;
19     ll sums = 0;
20     ll sumb = 0;
21     int n = 0;
22
23     int size_small() {
24         return small.size();
25     }
26
27     int size_big() {
28         return big.size();
29     }
30
31     void balance() {
32         while (size_small() > n / 2) {
33             int v = *small.rbegin();
34             small.erase(prev(small.end()));
35             big.insert(v);
36             sums -= v;
37             sumb += v;
38         }
39         while (size_big() > n - n / 2) {
40             int v = *big.begin();
41             big.erase(big.begin());
42             small.insert(v);
43             sumb -= v;
44             sums += v;
45         }
46     }
47
48     void add(int x) {
49         n++;
50         small.insert(x);
51         sums += x;

```

```

52 while (!small.empty() && *small.rbegin() > *big.
begin()) {
53     int v = *small.rbegin();
54     small.erase(prev(small.end()));
55     big.insert(v);
56     sums -= v;
57     sumb += v;
58 }
59 balance();
60 }
61
62 bool rem(int x) {
63     n--;
64     auto it1 = small.find(x);
65     auto it2 = big.find(x);
66     bool flag = false;
67     if (it1 != small.end()) {
68         sums -= *it1;
69         small.erase(it1);
70         flag = true;
71     } else if (it2 != big.end()) {
72         sumb -= *it2;
73         big.erase(it2);
74         flag = true;
75     }
76     balance();
77     return flag;
78 }
79
80 ll sum_small() {
81     return sums;
82 }
83
84 ll sum_big() {
85     return sumb;
86 }
87
88 int median() {
89     return *big.begin();
90 }
91 };

```

2.13 Priority Queue

```

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

```

3 Template

3.1 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 }

```

3.2 Template Clean

```

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

```

4 Graphs

4.1 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 = 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 }

```

4.2 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 }

```

4.3 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: ";
46         for (int v : cycle)
47             cout << v << " ";
48         cout << endl;
49     }
50 }

```

4.4 Ford Fulkerson Edmonds Karp

```

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

```

```

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

```

4.5 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 }

```

4.6 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 }

```

4.7 Dinic

```

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

```

```

31
32 using namespace std;
33
34 const ll INF = 1e18+10;
35
36 struct Edge {
37     int from;
38     int to;
39     ll capacity;
40     ll flow;
41     Edge* residual;
42
43     Edge() {}
44
45     Edge(int from, int to, ll capacity) : from(from), to(to), capacity(capacity) {
46         flow = 0;
47     }
48
49     ll get_capacity() {
50         return capacity - flow;
51     }
52
53     ll get_flow() {
54         return flow;
55     }
56
57     void augment(ll bottleneck) {
58         flow += bottleneck;
59         residual->flow -= bottleneck;
60     }
61
62     void reverse(ll bottleneck) {
63         flow -= bottleneck;
64         residual->flow += bottleneck;
65     }
66
67     bool operator<(const Edge& e) const {
68         return true;
69     }
70 };
71
72 struct Dinic {
73     int source;
74     int sink;
75     int nodes;
76     ll flow;
77     vector<vector<Edge*>> adj;
78     vector<int> level;
79     vector<int> next;
80     vector<int> reach;
81     vector<bool> visited;
82     vector<vector<int>> path;
83
84     Dinic(int source, int sink, int nodes) : source(source), sink(sink), nodes(nodes) {
85         adj.resize(nodes + 1);
86     }
87
88     void add_edge(int from, int to, ll capacity) {
89         Edge* e1 = new Edge(from, to, capacity);
90         Edge* e2 = new Edge(to, from, 0);
91         // Edge* e2 = new Edge(to, from, capacity);
92         e1->residual = e2;
93         e2->residual = e1;
94         adj[from].pb(e1);
95         adj[to].pb(e2);
96     }
97
98     bool bfs() {
99         level.assign(nodes + 1, -1);
100         queue<int> q;
101         q.push(source);
102
103         level[source] = 0;
104
105         while (!q.empty()) {
106             int node = q.front();
107             q.pop();
108
109             for (auto e : adj[node]) {
110                 if (level[e->to] == -1 && e->get_capacity() > 0) {
111                     level[e->to] = level[e->from] + 1;
112                     q.push(e->to);
113                 }
114             }
115         }
116
117         return level[sink] != -1;
118     }
119
120     ll dfs(int v, ll flow) {
121         if (v == sink)
122             return flow;
123
124         int sz = adj[v].size();
125         for (int i = next[v]; i < sz; i++) {
126             Edge* e = adj[v][i];
127             if (level[e->to] == level[e->from] + 1 && e->get_capacity() > 0) {
128                 ll bottleneck = dfs(e->to, min(flow, e->get_capacity()));
129                 if (bottleneck > 0) {
130                     e->augment(bottleneck);
131                     return bottleneck;
132                 }
133             }
134             next[v] = i + 1;
135         }
136
137         return 0;
138     }
139
140     ll max_flow() {
141         flow = 0;
142         while(bfs()) {
143             next.assign(nodes + 1, 0);
144             ll sent = -1;
145             while (sent != 0) {
146                 sent = dfs(source, INF);
147                 flow += sent;
148             }
149         }
150         return flow;
151     }
152
153     void reachable(int v) {
154         visited[v] = true;
155
156         for (auto e : adj[v]) {
157             if (!visited[e->to] && e->get_capacity() > 0) {
158                 reach.pb(e->to);
159                 visited[e->to] = true;
160                 reachable(e->to);
161             }
162         }
163     }
164
165     void print_min_cut() {
166         reach.clear();
167         visited.assign(nodes + 1, false);
168         reach.pb(source);
169         reachable(source);

```



```

170     for (auto v : reach) {
171         for (auto e : adj[v]) {
172             if (!visited[e->to] && e->
173 get_capacity() == 0) {
174                 cout << e->from << ' ' << e->to
175 << '\n';
176             }
177         }
178     }
179
180 ll build_path(int v, int id, ll flow) {
181     visited[v] = true;
182     if (v == sink) {
183         return flow;
184     }
185
186     for (auto e : adj[v]) {
187         if (!visited[e->to] && e->get_flow() > 0)
188     {
189         visited[e->to] = true;
190         ll bottleneck = build_path(e->to, id,
191 min(flow, e->get_flow()));
192         if (bottleneck > 0) {
193             path[id].pb(e->to);
194             e->reverse(bottleneck);
195             return bottleneck;
196         }
197     }
198     return 0;
199 }
200
201 void print_flow_path() {
202     path.clear();
203     ll sent = -1;
204     int id = -1;
205     while (sent != 0) {
206         visited.assign(nodes + 1, false);
207         path.pb(vector<int>{});
208         sent = build_path(source, ++id, INF);
209         path[id].pb(source);
210     }
211     path.pop_back();
212
213     for (int i = 0; i < id; i++) {
214         cout << path[i].size() << '\n';
215         reverse(path[i].begin(), path[i].end());
216         for (auto e : path[i]) {
217             cout << e << ' ';
218         }
219         cout << '\n';
220     }
221 }
222 };
223
224 int main() {
225     ios::sync_with_stdio(false);
226     cin.tie(NULL);
227
228     int n, m; cin >> n >> m;
229
230     Dinic dinic = Dinic(1, n, n);
231
232     for (int i = 1; i <= m; i++) {
233         int v, u; cin >> v >> u;
234         dinic.add_edge(v, u, 1);
235     }
236
237     cout << dinic.max_flow() << '\n';
238     // dinic.print_min_cut();

```

```

239     // dinic.print_flow_path();
240
241     return 0;
242 }

```

4.8 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 }

```

4.9 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
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 }

```

4.10 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 to : adj[v]) {
24         if (to == p) continue;
25         if (visited[to]) {
26             low[v] = min(low[v], tin[to]);
27         } else {
28             dfs(to, v);
29             low[v] = min(low[v], low[to]);
30             if (low[to] > tin[v]) {
31                 IS_BRIDGE(v, to);
32             }
33         }
34     }
35 }
36
37 void find_bridges() {
38     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 }

```

4.11 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
7 // Problem:
8 // https://cses.fi/problemset/task/2079/
9
10 // Complexity:
11 // O(n)
12
13 // How to use:
14 // get_subtree_size(0);
15 // cout << get_centroid(0) + 1 << endl;
16
17 int n;
18 vector<int> adj[MAX];
19 int subtree_size[MAX];
20
21 int get_subtree_size(int node, int par = -1) {
22     int &res = subtree_size[node];
23     res = 1;
24     for (int i : adj[node]) {
25         if (i == par) continue;
26         res += get_subtree_size(i, node);
27     }
28     return res;
29 }
30
31 int get_centroid(int node, int par = -1) {
32     for (int i : adj[node]) {
33         if (i == par) continue;
34         if (subtree_size[i] * 2 > n) { return
35             get_centroid(i, node); }
36     }
37     return node;
38 }
39
40 int main() {
41     cin >> n;
42     for (int i = 0; i < n - 1; i++) {
43         int u, v; cin >> u >> v;
44         u--; v--;
45         adj[u].push_back(v);
46         adj[v].push_back(u);
47     }
48
49     get_subtree_size(0);
50     cout << get_centroid(0) + 1 << endl;
51 }

```

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

```

4.13 2sat

```

1 // Description:
2 // Solves expression of the type (a v b) ^ (c v d) ^
3 // (e v f)
4 // Problem:
5 // https://cses.fi/problemset/task/1684
6 // Complexity:
7 // O(n + m) where n is the number of variables and m
8 // is the number of clauses
9
10 #include <bits/stdc++.h>

```

```

11 #define pb push_back
12 #define mp make_pair
13 #define pii pair<int, int>
14 #define ff first
15 #define ss second
16
17 using namespace std;
18
19 struct SAT {
20     int nodes;
21     int curr = 0;
22     int component = 0;
23     vector<vector<int>> adj;
24     vector<vector<int>> rev;
25     vector<vector<int>> condensed;
26     vector<pii> departure;
27     vector<bool> visited;
28     vector<int> scc;
29     vector<int> order;
30
31     // 1 to nodes
32     // nodes + 1 to 2 * nodes
33     SAT(int nodes) : nodes(nodes) {
34         adj.resize(2 * nodes + 1);
35         rev.resize(2 * nodes + 1);
36         visited.resize(2 * nodes + 1);
37         scc.resize(2 * nodes + 1);
38     }
39
40     void add_imp(int a, int b) {
41         adj[a].pb(b);
42         rev[b].pb(a);
43     }
44
45     int get_not(int a) {
46         if (a > nodes) return a - nodes;
47         return a + nodes;
48     }
49
50     void add_or(int a, int b) {
51         add_imp(get_not(a), b);
52         add_imp(get_not(b), a);
53     }
54
55     void add_nor(int a, int b) {
56         add_or(get_not(a), get_not(b));
57     }
58
59     void add_and(int a, int b) {
60         add_or(get_not(a), b);
61         add_or(a, get_not(b));
62         add_or(a, b);
63     }
64
65     void add_nand(int a, int b) {
66         add_or(get_not(a), b);
67         add_or(a, get_not(b));
68         add_or(get_not(a), get_not(b));
69     }
70
71     void add_xor(int a, int b) {
72         add_or(a, b);
73         add_or(get_not(a), get_not(b));
74     }
75
76     void add_xnor(int a, int b) {
77         add_or(get_not(a), b);
78         add_or(a, get_not(b));
79     }
80
81     void departure_time(int v) {
82         visited[v] = true;
83     }

```

```

84     for (auto u : adj[v]) {
85         if (!visited[u]) departure_time(u);
86     }
87
88     departure.pb(mp(++curr, v));
89 }
90
91 void find_component(int v, int component) {
92     scc[v] = component;
93     visited[v] = true;
94
95     for (auto u : rev[v]) {
96         if (!visited[u]) find_component(u,
97 component);
98     }
99
100 void topological_order(int v) {
101     visited[v] = true;
102
103     for (auto u : condensed[v]) {
104         if (!visited[u]) topological_order(u);
105     }
106
107     order.pb(v);
108 }
109
110 bool is_possible() {
111     component = 0;
112     for (int i = 1; i <= 2 * nodes; i++) {
113         if (!visited[i]) departure_time(i);
114     }
115
116     sort(departure.begin(), departure.end(),
117 greater<pii>());
118
119     visited.assign(2 * nodes + 1, false);
120
121     for (auto [_ , node] : departure) {
122         if (!visited[node]) find_component(node,
123 ++component);
124     }
125
126     for (int i = 1; i <= nodes; i++) {
127         if (scc[i] == scc[i + nodes]) return
128 false;
129     }
130
131     return true;
132 }
133
134 int find_value(int e, vector<int> &ans) {
135     if (e > nodes && ans[e - nodes] != 2) return
136 !ans[e - nodes];
137     if (e <= nodes && ans[e + nodes] != 2) return
138 !ans[e + nodes];
139     return 0;
140 }
141
142 vector<int> find_ans() {
143     condensed.resize(component + 1);
144
145     for (int i = 1; i <= 2 * nodes; i++) {
146         for (auto u : adj[i]) {
147             if (scc[i] != scc[u]) condensed[scc[i]
148 ]].pb(scc[u]);
149         }
150     }
151
152     visited.assign(component + 1, false);
153
154     for (int i = 1; i <= component; i++) {
155         if (!visited[i]) topological_order(i);

```

```

150     }
151
152     reverse(order.begin(), order.end());
153
154     // 0 - false
155     // 1 - true
156     // 2 - no value yet
157     vector<int> ans(2 * nodes + 1, 2);
158
159     vector<vector<int>> belong(component + 1);
160
161     for (int i = 1; i <= 2 * nodes; i++) {
162         belong[scc[i]].pb(i);
163     }
164
165     for (auto p : order) {
166         for (auto e : belong[p]) {
167             ans[e] = find_value(e, ans);
168         }
169     }
170
171     return ans;
172 }
173 };
174
175 int main() {
176     ios::sync_with_stdio(false);
177     cin.tie(NULL);
178
179     int n, m; cin >> n >> m;
180
181     SAT sat = SAT(m);
182
183     for (int i = 0; i < n; i++) {
184         char op1, op2; int a, b; cin >> op1 >> a >>
185 op2 >> b;
186         if (op1 == '+' && op2 == '+') sat.add_or(a, b
187 );
188         if (op1 == '-' && op2 == '-') sat.add_or(sat.
189 get_not(a), sat.get_not(b));
190         if (op1 == '+' && op2 == '-') sat.add_or(a,
191 sat.get_not(b));
192         if (op1 == '-' && op2 == '+') sat.add_or(sat.
193 get_not(a), b);
194     }
195
196     if (!sat.is_possible()) cout << "IMPOSSIBLE\n";
197     else {
198         vector<int> ans = sat.find_ans();
199         for (int i = 1; i <= m; i++) {
200             cout << (ans[i] == 1 ? '+' : '-') << ' ';
201         }
202         cout << '\n';
203     }
204
205     return 0;
206 }

```

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

```

4.15 Kruskall

```

1 struct DSU {
2     int n;
3     vector<int> link, sizes;
4
5     DSU(int n) {
6         this->n = n;
7         link.assign(n+1, 0);
8         sizes.assign(n+1, 1);
9
10        for (int i = 0; i <= n; i++)
11            link[i] = i;
12    }
13
14    int find(int x) {
15        while (x != link[x])
16            x = link[x];
17
18        return x;
19    }
20
21    bool same(int a, int b) {
22        return find(a) == find(b);
23    }
24
25    void unite(int a, int b) {
26        a = find(a);
27        b = find(b);
28
29        if (a == b) return;
30
31        if (sizes[a] < sizes[b])
32            swap(a, b);
33
34        sizes[a] += sizes[b];
35        link[b] = a;
36    }
37 };
38
39 struct Edge {
40     int u, v;
41     long long weight;
42

```

```

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

```

4.16 Lca

```

1 // Description:
2 // Find the lowest common ancestor between two nodes
3 // in a tree
4
5 // Problem:
6 // https://cses.fi/problemset/task/1135
7
8 // Complexity:
9 // O(log n)
10
11 // How to use:
12 // preprocess();
13 // lca(a, b);
14
15 // Notes
16 // To calculate the distance between two nodes use
17 // the following formula
18 // level_peso[a] + level_peso[b] - 2*level_peso[lca(a,
19 // b)]
20
21 const int MAX = 2e5+10;
22 const int BITS = 30;
23
24 vector<pii> adj[MAX];
25 vector<bool> visited(MAX);
26
27 int up[MAX][BITS + 1];
28 int level[MAX];
29 int level_peso[MAX];
30
31 void find_level() {
32     queue<pii> q;
33
34     q.push(mp(1, 0));
35     visited[1] = true;
36
37     while (!q.empty()) {
38         auto [v, depth] = q.front();
39

```

```

36     q.pop();
37     level[v] = depth;
38
39     for (auto [u,d] : adj[v]) {
40         if (!visited[u]) {
41             visited[u] = true;
42             up[u][0] = v;
43             q.push(mp(u, depth + 1));
44         }
45     }
46 }
47 }
48
49 void find_level_peso() {
50     queue<pii> q;
51
52     q.push(mp(1, 0));
53     visited[1] = true;
54
55     while (!q.empty()) {
56         auto [v, depth] = q.front();
57         q.pop();
58         level_peso[v] = depth;
59
60         for (auto [u,d] : adj[v]) {
61             if (!visited[u]) {
62                 visited[u] = true;
63                 up[u][0] = v;
64                 q.push(mp(u, depth + d));
65             }
66         }
67     }
68 }
69
70 int lca(int a, int b) {
71     // get the nodes to the same level
72     int mn = min(level[a], level[b]);
73
74     for (int j = 0; j <= BITS; j++) {
75         if (a != -1 && ((level[a] - mn) & (1 << j))) a
76         = up[a][j];
77         if (b != -1 && ((level[b] - mn) & (1 << j))) b
78         = up[b][j];
79     }
80
81     // special case
82     if (a == b) return a;
83
84     // binary search
85     for (int j = BITS; j >= 0; j--) {
86         if (up[a][j] != up[b][j]) {
87             a = up[a][j];
88             b = up[b][j];
89         }
90     }
91     return up[a][0];
92 }
93
94 void preprocess() {
95     visited = vector<bool>(MAX, false);
96     find_level();
97     visited = vector<bool>(MAX, false);
98     find_level_peso();
99
100    for (int j = 1; j <= BITS; j++) {
101        for (int i = 1; i <= n; i++) {
102            if (up[i][j - 1] != -1) up[i][j] = up[up[i][j - 1]][j - 1];
103        }
104    }
105 }

```

4.17 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     solve(1, 'A');
73
74     if (!flag) cout << "Impossible!\n";
75     else {
76         for (int i = 1; i <= n; i++) {
77             cout << ans[i] << ' ';
78         }
79         cout << '\n';
80     }
81 }
82
83 return 0;
84 }

```

5 Strings

5.1 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 }

```

5.2 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     }
39 }

```

```

35     }
36
37     else if (LCS_table[i - 1][j] > LCS_table[i][j -
38         1])
39         i--;
40     else
41         j--;
42 }
43 return lcsAlgo;
44 }

```

5.3 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 }

```

5.4 Generate All Sequences Length K

```

1 // gera todas as ípossveis âsequencias 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    }
16
17    return ans;
18 }

```

5.5 Trie

```

1 const int K = 26;
2
3 struct Vertex {
4     int next[K];
5     bool output = false;
6     int p = -1;
7     char pch;
8     int link = -1;
9     int go[K];
10
11     Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
12         fill(begin(next), end(next), -1);
13         fill(begin(go), end(go), -1);
14     }
15 };
16
17 vector<Vertex> t(1);
18
19 void add_string(string const& s) {
20     int v = 0;

```

```

21     for (char ch : s) {
22         int c = ch - 'a';
23         if (t[v].next[c] == -1) {
24             t[v].next[c] = t.size();
25             t.emplace_back(v, ch);
26         }
27         v = t[v].next[c];
28     }
29     t[v].output = true;
30 }
31
32 int go(int v, char ch);
33
34 int get_link(int v) {
35     if (t[v].link == -1) {
36         if (v == 0 || t[v].p == 0)
37             t[v].link = 0;
38         else
39             t[v].link = go(get_link(t[v].p), t[v].pch);
40     }
41     return t[v].link;
42 }
43
44 int go(int v, char ch) {
45     int c = ch - 'a';
46     if (t[v].go[c] == -1) {
47         if (t[v].next[c] != -1)
48             t[v].go[c] = t[v].next[c];
49         else
50             t[v].go[c] = v == 0 ? 0 : go(get_link(v),
51                                         ch);
52     }
53     return t[v].go[c];
54 }

```

5.6 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              ++z[i];
9          if (i + z[i] - 1 > r)
10             l = i, r = i + z[i] - 1;
11     }
12     return z;
13 }

```

6 DP

6.1 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     }

```

```

14     return dp[n][m];
15 }

```

6.2 Substr Palindrome

```

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

```

6.3 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 }
```

6.4 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
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     }
```

6.5 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    return tabela[k] = melhor;
18 }
```

6.6 Digits

```
1 // achar a quantidade de numeros menores que R que
2 // possuem no maximo 3 digitos nao nulos
3 // a ideia eh utilizar da ordem lexicografica para
4 // checar isso pois se temos por exemplo
5 // o numero 8500, a gente sabe que se pegarmos o
6 // numero 7... qualquer digito depois do 7
7 // sera necessariamente menor q 8500
8
9 string r;
10 int tab[20][2][5];
11
12 // i - digito de R
13 // menor - ja pegou um numero menor que um digito de
14 // R
15 // qt - quantidade de digitos nao nulos
16 int dp(int i, bool menor, int qt){
17     if(qt > 3) return 0;
18     if(i >= r.size()) return 1;
19     if(tab[i][menor][qt] != -1) return tab[i][menor][
20     qt];
21
22     int dr = r[i] - '0';
23     int res = 0;
24
25     for(int d = 0; d <= 9; d++) {
26         int dnn = qt + (d > 0);
27         if(menor == true) {
28             res += dp(i+1, true, dnn);
29         }
30         else if(d < dr) {
31             res += dp(i+1, true, dnn);
32         }
33         else if(d == dr) {
34             res += dp(i+1, false, dnn);
35         }
36     }
37
38     return tab[i][menor][qt] = res;
39 }
```

6.7 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 }
```

6.8 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 ão]
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 }

```

7 Math

7.1 Function Root

```

1 const ld EPS1 = 1e-9; // iteration precision error
2 const ld EPS2 = 1e-4; // output precision error
3
4 ld f(ld x) {
5     // exp(-x) == e^(-x)
6     return p * exp(-x) + q * sin(x) + r * cos(x) + s *
7     tan(x) + t * x * x + u;
8 }
9
10 ld root(ld a, ld b) {
11     while (b - a >= EPS1) {
12         ld c = (a + b) / 2.0;
13         ld y = f(c);
14
15         if (y < 0) b = c;
16         else a = c;

```

```

16     }
17
18     return (a + b) / 2;
19 }
20
21 int main() {
22     ld ans = root(0, 1);
23     if (abs(f(ans)) <= EPS2) cout << fixed <<
24         setprecision(4) << ans << '\n';
25     else cout << "No solution\n";
26
27     return 0;
28 }

```

7.2 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 }

```

7.3 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 }

```

7.4 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 }

```

7.5 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 }

```

7.6 Horner Algorithm

```

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

```

7.7 Sieve Of Eratosthenes

```

1 vector<bool> is_prime(MAX, true);
2 vector<int> primes;
3
4 void sieve() {
5     is_prime[0] = is_prime[1] = false;

```

```

6     for (int i = 2; i < MAX; i++) {
7         if (is_prime[i]) {
8             primes.push_back(i);
9
10            for (int j = i + i; j < MAX; j += i)
11                is_prime[j] = false;
12        }
13    }
14 }

```

7.8 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 }

```

7.9 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 }

```

7.10 Ceil

```

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

```

7.11 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
13 // (2 x 1) = (2 x 2) * (2 x 1)
14 // F(n)      = a b * F(n - 1)
15 // F(n - 1)   c d   F(n - 2)
16
17 // Another Example:
18 // Given a grid 3 x n, you want to color it using 3
19 // distinct colors so that
20 // no adjacent place has the same color. In how many
21 // different ways can you do that?
22 // There are 6 ways for the first column to be
23 // colored using 3 distinct colors

```

```

20 // ans 6 ways using 2 equal colors and 1 distinct one
21
22 // Adding another column, there are:
23 // 3 ways to go from 2 equal to 2 equal
24 // 2 ways to go from 2 equal to 3 distinct
25 // 2 ways to go from 3 distinct to 2 equal
26 // 2 ways to go from 3 distinct to 3 distinct
27
28 // So we star with matrix 6 6 and multiply it by the
    transition 3 2 and get 18 12
29 //
        2 2      12 12
30 // the we can exponentiate this matrix to find the
    nth column
31
32 // Problem:
33 // https://cses.fi/problemset/task/1722/
34
35 // Complexity:
36 // O(log n)
37
38 // How to use:
39 // vector<vector<ll>> v = {{1, 1}, {1, 0}};
40 // Matriz transition = Matriz(v);
41 // cout << fexp(transition, n)[0][1] << '\n';
42
43 using ll = long long;
44
45 const int MOD = 1e9+7;
46
47 struct Matriz{
48     vector<vector<ll>> mat;
49     int rows, columns;
50
51     vector<ll> operator[](int i){
52         return mat[i];
53     }
54
55     Matriz(vector<vector<ll>>& matriz){
56         mat = matriz;
57         rows = mat.size();
58         columns = mat[0].size();
59     }
60
61     Matriz(int row, int column, bool identity=false){
62         rows = row; columns = column;
63         mat.assign(rows, vector<ll>(columns, 0));
64         if(identity) {
65             for(int i = 0; i < min(rows, columns); i
66 ++) {
67                 mat[i][i] = 1;
68             }
69         }
70
71     Matriz operator * (Matriz a) {
72         assert(columns == a.rows);
73         vector<vector<ll>> resp(rows, vector<ll>(a.
74 columns, 0));
75
76         for(int i = 0; i < rows; i++){
77             for(int j = 0; j < a.columns; j++){
78                 for(int k = 0; k < a.rows; k++){
79                     resp[i][j] = (resp[i][j] + (mat[i
80 ][k] * 1LL * a[k][j]) % MOD) % MOD;
81                 }
82             }
83         }
84         return Matriz(resp);
85     }
86
87     Matriz operator + (Matriz a) {
88         assert(rows == a.rows && columns == a.columns

```

```

87 );
88     vector<vector<ll>> resp(rows, vector<ll>(
89 columns,0));
90     for(int i = 0; i < rows; i++){
91         for(int j = 0; j < columns; j++){
92             resp[i][j] = (resp[i][j] + mat[i][j]
93 + a[i][j]) % MOD;
94         }
95     }
96     return Matriz(resp);
97 }
98
99 };
100
101 Matriz fexp(Matriz base, ll exponent){
102     Matriz result = Matriz(base.rows, base.rows, 1);
103     while(exponent > 0){
104         if(exponent & 1LL) result = result * base;
105         base = base * base;
106         exponent = exponent >> 1;
107     }
108     return result;
109 }

```

7.12 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
    /99999/106
24
25 // consider trivial case a or b is 0
26 int gcd(int a, int b, int& x, int& y) {
27     if (b == 0) {
28         x = 1;
29         y = 0;
30         return a;
31     }
32     int x1, y1;
33     int d = gcd(b, a % b, x1, y1);
34     x = y1;
35     y = x1 - y1 * (a / b);
36     return d;
37 }
38
39 // x and y are one solution and g is the gcd, all
    passed as reference
40 // minx <= x <= maxx miny <= y <= maxy
41 bool find_any_solution(int a, int b, int c, int &x0,
    int &y0, int &g) {

```

```

42     g = gcd(abs(a), abs(b), x0, y0);
43     if (c % g) {
44         return false;
45     }
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
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);
91     if (y > maxy)
92         shift_solution(x, y, a, b, sign_a);
93     int rx2 = x;
94
95     if (lx2 > rx2)
96         swap(lx2, rx2);
97     int lx = max(lx1, lx2);
98     int rx = min(rx1, rx2);
99
100     if (lx > rx)
101         return 0;
102     return (rx - lx) / abs(b) + 1;

```

7.13 Representation Arbitrary Base

```

1 const string digits { "0123456789
2     ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
3
4 string representation(int n, int b) {
5     string rep;
6     do {

```

```

7         rep.push_back(digits[n % b]);
8         n /= b;
9     } while (n);
10
11     reverse(rep.begin(), rep.end());
12
13     return rep;
14 }

```

7.14 To Decimal

```

1 const string digits { "0123456789
2     ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
3
4 long long to_decimal(const string& rep, long long
5     base) {
6     long long n = 0;
7
8     for (auto c : rep) {
9         n *= base;
10        n += digits.find(c);
11    }
12    return n;

```

7.15 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 }

```

8 Algorithms

8.1 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 }

```

8.2 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         //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 }

```

8.3 Binary Search First True

```

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

```

8.4 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 }

```

8.5 Binary Search Last True

```

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

```

```

11    return lo;
12 }

```

8.6 Biggest K

```

1 // Description: Gets sum of k biggest or k smallest
elements in an array
2
3 // Problem: https://atcoder.jp/contests/abc306/tasks/
abc306_e
4
5 // Complexity: O(log n)
6
7 struct SetSum {
8     ll s = 0;
9     multiset<ll> mt;
10    void add(ll x){
11        mt.insert(x);
12        s += x;
13    }
14    int pop(ll x){
15        auto f = mt.find(x);
16        if(f == mt.end()) return 0;
17        mt.erase(f);
18        s -= x;
19        return 1;
20    }
21 };
22
23 struct BigK {
24     int k;
25     SetSum gt, mt;
26     BigK(int _k){
27         k = _k;
28     }
29     void balancear(){
30         while((int)gt.mt.size() < k && (int)mt.mt.
size()){
31             auto p = (prev(mt.mt.end()));
32             gt.add(*p);
33             mt.pop(*p);
34         }
35         while((int)mt.mt.size() && (int)gt.mt.size()
&&
36             *(gt.mt.begin()) < *(prev(mt.mt.end())) ){
37             ll u = *(gt.mt.begin());
38             ll v = *(prev(mt.mt.end()));
39             gt.pop(u); mt.pop(v);
40             gt.add(v); mt.add(u);
41         }
42     }
43     void add(ll x){
44         mt.add(x);
45         balancear();
46     }
47     void rem(ll x){
48         //x = -x;
49         if(mt.pop(x) == 0)
50             gt.pop(x);
51         balancear();
52     }
53 };
54
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(NULL);
58
59     int n, k, q; cin >> n >> k >> q;
60
61     BigK big = BigK(k);
62
63     int arr[n] = {};
64

```

```
65 while (q--) {
66     int pos, num; cin >> pos >> num;
67     pos--;
68     big.rem(arr[pos]);
69     arr[pos] = num;
70     big.add(arr[pos]);
```

```
71
72     cout << big.gt.s << '\n';
73 }
74
75 return 0;
76 }
```