Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1 Misc

1.1 Split

```cpp
vector<string> split(string txt, char key = ' '){
    vector<string> ans;

    string palTemp = "";
    for(int i = 0; i < txt.size(); i++){

        if(txt[i] == key){
            if(palTemp.size() > 0){
                ans.push_back(palTemp);
                palTemp = "";
            }
        } else{
            palTemp += txt[i];
        }

    }

    if(palTemp.size() > 0)
        ans.push_back(palTemp);

    return ans;
}
```

1.2 Int128

```cpp
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
```

2 Geometry

3 Data Structures

3.1 Range Query Point Update

```cpp
// Description:
// Indexed at zero
// Query - get sum of elements from range (l, r)
    inclusive
// Update - update element at position id to a value
    val

// Problem:
// https://codeforces.com/edu/course/2/lesson/4/1/
    practice/contest/273169/problem/B

// Complexity:
```

```cpp
// O(log n) for both query and update

// How to use:
// Segtree seg = Segtree(n);
// seg.build(v);

// Notes
// Change neutral element and f function to perform a
    different operation

// If you want to change the operations to point
    query and range update
// Use the same segtree, but perform the following
    operations
// Query - seg.query(0, id);
// Update - seg.update(l, v); seg.update(r + 1, -v);

typedef long long ftype;

struct Segtree {
    vector<ftype> seg;
    int n;
    const ftype NEUTRAL = 0;

    Segtree(int n) {
        int sz = 1;
        while (sz < n) sz *= 2;
        this->n = sz;

        seg.assign(2*sz, NEUTRAL);
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype query(int pos, int ini, int fim, int p, int
     q) {
        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m +
     1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int id,
     int val) {
        if (ini > id || fim < id) {
            return;
        }

        if (ini == id && fim == id) {
            seg[pos] = val;

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, id, val);
        update(d, m + 1, fim, id, val);
```

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

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

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

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

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

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

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

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

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

3.2 Minimum And Amount

```
// Description:
// Query - get minimum element in a range (l, r)
    inclusive
// and also the number of times it appears in that
    range
// Update - update element at position id to a value
    val

// Problem:
// https://codeforces.com/edu/course/2/lesson/4/1/
    practice/contest/273169/problem/C

// Complexity:
// O(log n) for both query and update

// How to use:
// Segtree seg = Segtree(n);
// seg.build(v);

#define pii pair<int, int>
#define mp make_pair
#define ff first
#define ss second

const int INF = 1e9+17;

typedef pii ftype;

struct Segtree {
    vector<ftype> seg;
    int n;
    const ftype NEUTRAL = mp(INF, 0);

    Segtree(int n) {
        int sz = 1;
        while (sz < n) sz *= 2;
        this->n = sz;

        seg.assign(2*sz, NEUTRAL);
    }

    ftype f(ftype a, ftype b) {
        if (a.ff < b.ff) return a;
        if (b.ff < a.ff) return b;

        return mp(a.ff, a.ss + b.ss);
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int id, int val) {
        if (ini > id || fim < id) {
            return;
        }

        if (ini == id && fim == id) {
            seg[pos] = mp(val, 1);

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, id, val);
        update(d, m + 1, fim, id, val);

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

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            if (ini < (int)v.size()) {
                seg[pos] = mp(v[ini], 1);
            }
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        build(e, ini, m, v);
```

3.3 Dynamic Implicit Sparse

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

3.3 Dynamic Implicit Sparse

```cpp
1 // Description:
2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
       stored in an array
5 // and the queries need to be answered online so we
       can't sort the nodes and compress them
6 // we create nodes only when they are needed so there
      'll be (Q*log(MAX)) nodes
7 // where Q is the number of queries and MAX is the
      maximum index a node can assume
8
9 // Query - get sum of elements from range (l, r)
      inclusive
10 // Update - update element at position id to a value
      val
11
12 // Problem:
13 // https://cses.fi/problemset/task/1648
14
15 // Complexity:
16 // O(log n) for both query and update
17
18 // How to use:
19 // MAX is the maximum index a node can assume
20 // Create a default null node
21 // Create a node to be the root of the segtree
22
23 // Segtree seg = Segtree(MAX);
24 // seg.create();
25 // seg.create();
26
27 typedef long long ftype;
28
29 const int MAX = 1e9+17;
30
31 struct Segtree {
32     vector<ftype> seg, d, e, lazy;
33     const ftype NEUTRAL = 0;
34     int n;
35
36     Segtree(int n) {
37         this->n = n;
38     }
39
40     ftype f(ftype a, ftype b) {
41        return a + b;
42    }
43
44    ftype create() {
45        seg.push_back(0);
46        e.push_back(0);
47        d.push_back(0);
48        return seg.size() - 1;
49    }
50
51    ftype query(int pos, int ini, int fim, int p, int
 q) {
52        if (q < ini || p > fim) return NEUTRAL;
53        if (pos == 0) return 0;
54        if (p <= ini && fim <= q) return seg[pos];
55        int m = (ini + fim) >> 1;
56        return f(query(e[pos], ini, m, p, q), query(d
[pos], m + 1, fim, p, q));
57    }
58
59    void update(int pos, int ini, int fim, int id,
int val) {
60        if (ini > id || fim < id) {
61            return;
62        }
63
64        if (ini == fim) {
65            seg[pos] = val;
66
67            return;
68        }
69
70        int m = (ini + fim) >> 1;
71
72        if (id <= m) {
73            if (e[pos] == 0) e[pos] = create();
74            update(e[pos], ini, m, id, val);
75        } else {
76            if (d[pos] == 0) d[pos] = create();
77            update(d[pos], m + 1, fim, id, val);
78        }
79
80        seg[pos] = f(seg[e[pos]], seg[d[pos]]);
81    }
82
83    ftype query(int p, int q) {
84        return query(1, 1, n, p, q);
85    }
86
87    void update(int id, int val) {
88        update(1, 1, n, id, val);
89    }
90 };
```

3.4 Lazy Dynamic Implicit Sparse

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

```
12  // Problem:
13  // https://oj.uz/problem/view/IZhO12_apple
14
15  // Complexity:
16  // O(log n) for both query and update
17
18  // How to use:
19  // MAX is the maximum index a node can assume
20  // Create a default null node
21  // Create a node to be the root of the segtree
22
23  // Segtree seg = Segtree(MAX);
24  // seg.create();
25  // seg.create();
26
27  typedef long long ftype;
28
29  const int MAX = 1e9+17;
30
31  typedef long long ftype;
32
33  const int MAX = 1e9+17;
34
35  struct Segtree {
36      vector<ftype> seg, d, e, lazy;
37      const ftype NEUTRAL = 0;
38      const ftype NEUTRAL_LAZY = -1;
39      int n;
40
41      Segtree(int n) {
42          this->n = n;
43      }
44
45      ftype apply_lazy(ftype a, ftype b, int len) {
46          if (b == NEUTRAL_LAZY) return a;
47          else return b * len;
48      }
49
50      void propagate(int pos, int ini, int fim) {
51          if (seg[pos] == 0) return;
52
53          if (ini == fim) {
54              return;
55          }
56
57          int m = (ini + fim) >> 1;
58
59          if (e[pos] == 0) e[pos] = create();
60          if (d[pos] == 0) d[pos] = create();
61
62          lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[
pos], 1);
63          lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[
pos], 1);
64
65          seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[
pos], m - ini + 1);
66          seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[
pos], fim - m);
67
68          lazy[pos] = NEUTRAL_LAZY;
69      }
70
71      ftype f(ftype a, ftype b) {
72          return a + b;
73      }
74
75      ftype create() {
76          seg.push_back(0);
77          e.push_back(0);
78          d.push_back(0);
79          lazy.push_back(-1);
80          return seg.size() - 1;
```

```
81      }
82
83      ftype query(int pos, int ini, int fim, int p, int
 q) {
84          propagate(pos, ini, fim);
85          if (q < ini || p > fim) return NEUTRAL;
86          if (pos == 0) return 0;
87          if (p <= ini && fim <= q) return seg[pos];
88          int m = (ini + fim) >> 1;
89          return f(query(e[pos], ini, m, p, q), query(d
[pos], m + 1, fim, p, q));
90      }
91
92      void update(int pos, int ini, int fim, int p, int
 q, int val) {
93          propagate(pos, ini, fim);
94          if (ini > q || fim < p) {
95              return;
96          }
97
98          if (ini >= p && fim <= q) {
99              lazy[pos] = apply_lazy(lazy[pos], val, 1)
;
100             seg[pos] = apply_lazy(seg[pos], val, fim
- ini + 1);
101
102             return;
103         }
104
105         int m = (ini + fim) >> 1;
106
107         if (e[pos] == 0) e[pos] = create();
108         update(e[pos], ini, m, p, q, val);
109
110         if (d[pos] == 0) d[pos] = create();
111         update(d[pos], m + 1, fim, p, q, val);
112
113         seg[pos] = f(seg[e[pos]], seg[d[pos]]);
114     }
115
116     ftype query(int p, int q) {
117         return query(1, 1, n, p, q);
118     }
119
120     void update(int p, int q, int val) {
121         update(1, 1, n, p, q, val);
122     }
123 };
```

3.5   Lazy

```
1  // Description:
2  // Query - get sum of elements from range (l, r)
       inclusive
3  // Update - add a value val to elementos from range (
       l, r) inclusive
4
5  // Problem:
6  // https://codeforces.com/edu/course/2/lesson/5/1/
       practice/contest/279634/problem/A
7
8  // Complexity:
9  // O(log n) for both query and update
10
11  // How to use:
12  // Segtree seg = Segtree(n);
13  // seg.build(v);
14
15  // Notes
16  // Change neutral element and f function to perform a
        different operation
17
18  typedef long long ftype;
```

5

```cpp
struct Segtree {
    vector<ftype> seg;
    vector<ftype> lazy;
    int n;
    const ftype NEUTRAL = 0;
    const ftype NEUTRAL_LAZY = -1;

    Segtree(int n) {
        int sz = 1;
        while (sz < n) sz *= 2;
        this->n = sz;

        seg.assign(2*sz, NEUTRAL);
        lazy.assign(2*sz, NEUTRAL_LAZY);
    }

    ftype apply_lazy(ftype a, ftype b, int len) {
        if (b == NEUTRAL_LAZY) return a;
        if (a == NEUTRAL_LAZY) return b * len;
        else return a + b * len;
    }

    void propagate(int pos, int ini, int fim) {
        if (ini == fim) {
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
        lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);

        seg[e] = apply_lazy(seg[e], lazy[pos], m -
    ini + 1);
        seg[d] = apply_lazy(seg[d], lazy[pos], fim -
    m);

        lazy[pos] = NEUTRAL_LAZY;
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype query(int pos, int ini, int fim, int p, int
     q) {
        propagate(pos, ini, fim);

        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m +
     1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int p, int
     q, int val) {
        propagate(pos, ini, fim);

        if (ini > q || fim < p) {
            return;
        }
```

```cpp
        }

        if (ini >= p && fim <= q) {
            lazy[pos] = apply_lazy(lazy[pos], val, 1)
    ;
            seg[pos] = apply_lazy(seg[pos], val, fim
     - ini + 1);

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, p, q, val);
        update(d, m + 1, fim, p, q, val);

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

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

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

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

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

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

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

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

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

3.6 Segment With Maximum Sum

```cpp
// Description:
// Query - get sum of segment that is maximum among
    all segments
// E.g
// Array: 5 -4 4 3 -5
// Maximum segment sum: 8 because 5 + (-4) + 4 = 8
```

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

```cpp
70          }
71
72          int e = 2*pos + 1;
73          int d = 2*pos + 2;
74          int m = ini + (fim - ini) / 2;
75
76          return f(query(e, ini, m, p, q), query(d, m +
        1, fim, p, q));
77      }
78
79      void update(int pos, int ini, int fim, int id,
       int val) {
80          if (ini > id || fim < id) {
81              return;
82          }
83
84          if (ini == id && fim == id) {
85              seg[pos] = Node(val, val, val, val);
86
87              return;
88          }
89
90          int e = 2*pos + 1;
91          int d = 2*pos + 2;
92          int m = ini + (fim - ini) / 2;
93
94          update(e, ini, m, id, val);
95          update(d, m + 1, fim, id, val);
96
97          seg[pos] = f(seg[e], seg[d]);
98      }
99
100     void build(int pos, int ini, int fim, vector<int>
        &v) {
101         if (ini == fim) {
102             // se a çãposio existir no array original
103             // seg tamanho potencia de dois
104             if (ini < (int)v.size()) {
105                 seg[pos] = Node(v[ini], v[ini], v[ini
       ], v[ini]);
106             }
107             return;
108         }
109
110         int e = 2*pos + 1;
111         int d = 2*pos + 2;
112         int m = ini + (fim - ini) / 2;
113
114         build(e, ini, m, v);
115         build(d, m + 1, fim, v);
116
117         seg[pos] = f(seg[e], seg[d]);
118     }
119
120     ftype query(int p, int q) {
121         return query(0, 0, n - 1, p, q);
122     }
123
124     void update(int id, int val) {
125         update(0, 0, n - 1, id, val);
126     }
127
128     void build(vector<int> &v) {
129         build(0, 0, n - 1, v);
130     }
131
132     void debug() {
133         for (auto e : seg) {
134             cout << e.max_seg << ' ' << e.pref << ' '
        << e.suf << ' ' << e.sum << '\n';
135         }
136         cout << '\n';
137     }
```

```
138 };
```

3.7 Segtree2d

```cpp
1  // Description:
2  // Indexed at zero
3  // Given a N x M grid, where i represents the row and
       j the column, perform the following operations
4  // update(j, i) - update the value of grid[i][j]
5  // query(j1, j2, i1, i2) - return the sum of values
       inside the rectangle
6  // defined by grid[i1][j1] and grid[i2][j2] inclusive
7
8  // Problem:
9  // https://cses.fi/problemset/task/1739/
10
11 // Complexity:
12 // Time complexity:
13 // O(log N * log M) for both query and update
14 // O(N * M) for build
15 // Memory complexity:
16 // 4 * M * N
17
18 // How to use:
19 // Segtree2D seg = Segtree2D(n, n);
20 // vector<vector<int>> v(n, vector<int>(n));
21 // seg.build(v);
22
23 // Notes
24 // Indexed at zero
25
26 struct Segtree2D {
27     const int MAXN = 1025;
28     int N, M;
29
30     vector<vector<int>> seg;
31
32     Segtree2D(int N, int M) {
33         this->N = N;
34         this->M = M;
35         seg.resize(2*MAXN, vector<int>(2*MAXN));
36     }
37
38     void buildY(int noX, int lX, int rX, int noY, int
          lY, int rY, vector<vector<int>> &v){
39         if(lY == rY){
40             if(lX == rX){
41                 seg[noX][noY] = v[rX][rY];
42             }else{
43                 seg[noX][noY] = seg[2*noX+1][noY] +
    seg[2*noX+2][noY];
44             }
45         }else{
46             int m = (lY+rY)/2;
47
48             buildY(noX, lX, rX, 2*noY+1, lY, m, v);
49             buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);
50
51             seg[noX][noY] = seg[noX][2*noY+1] + seg[
    noX][2*noY+2];
52         }
53     }
54
55     void buildX(int noX, int lX, int rX, vector<
          vector<int>> &v){
56         if(lX != rX){
57             int m = (lX+rX)/2;
58
59             buildX(2*noX+1, lX, m, v);
60             buildX(2*noX+2, m+1, rX, v);
61         }
62
63         buildY(noX, lX, rX, 0, 0, M - 1, v);
64     }
65
66     void updateY(int noX, int lX, int rX, int noY,
          int lY, int rY, int y){
67         if(lY == rY){
68             if(lX == rX){
69                 seg[noX][noY] = !seg[noX][noY];
70             }else{
71                 seg[noX][noY] = seg[2*noX+1][noY] +
    seg[2*noX+2][noY];
72             }
73         }else{
74             int m = (lY+rY)/2;
75
76             if(y <= m){
77                 updateY(noX, lX, rX, 2*noY+1,lY, m, y
    );
78             }else if(m < y){
79                 updateY(noX, lX, rX, 2*noY+2, m+1, rY
    , y);
80             }
81
82             seg[noX][noY] = seg[noX][2*noY+1] + seg[
    noX][2*noY+2];
83         }
84     }
85
86     void updateX(int noX, int lX, int rX, int x, int
          y){
87         int m = (lX+rX)/2;
88
89         if(lX != rX){
90             if(x <= m){
91                 updateX(2*noX+1, lX, m, x, y);
92             }else if(m < x){
93                 updateX(2*noX+2, m+1, rX, x, y);
94             }
95         }
96
97         updateY(noX, lX, rX, 0, 0, M - 1, y);
98     }
99
100    int queryY(int noX, int noY, int lY, int rY, int
          aY, int bY){
101        if(aY <= lY && rY <= bY) return seg[noX][noY
    ];
102
103        int m = (lY+rY)/2;
104
105        if(bY <= m) return queryY(noX, 2*noY+1, lY, m
    , aY, bY);
106        if(m < aY) return queryY(noX, 2*noY+2, m+1,
    rY, aY, bY);
107
108        return queryY(noX, 2*noY+1, lY, m, aY, bY) +
    queryY(noX, 2*noY+2, m+1, rY, aY, bY);
109    }
110
111    int queryX(int noX, int lX, int rX, int aX, int
          bX, int aY, int bY){
112        if(aX <= lX && rX <= bX) return queryY(noX,
    0, 0, M - 1, aY, bY);
113
114        int m = (lX+rX)/2;
115
116        if(bX <= m) return queryX(2*noX+1, lX, m, aX,
    bX, aY, bY);
117        if(m < aX) return queryX(2*noX+2, m+1, rX, aX
    , bX, aY, bY);
118
119        return queryX(2*noX+1, lX, m, aX, bX, aY, bY)
    + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
120    }
```

```
121    void build(vector<vector<int>> &v) {
122        buildX(0, 0, N - 1, v);
123    }
124
125
126    int query(int aX, int bX, int aY, int bY) {
127        return queryX(0, 0, N - 1, aX, bX, aY, bY);
128    }
129
130    void update(int x, int y) {
131        updateX(0, 0, N - 1, x, y);
132    }
133 };
```

3.8 Persistent

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

```
50         if (pos == 0) return 0;
51         if (p <= ini && fim <= q) return seg[pos];
52         int m = (ini + fim) >> 1;
53         return f(query(e[pos], ini, m, p, q), query(d
[pos], m + 1, fim, p, q));
54     }
55
56     int update(int pos, int ini, int fim, int id, int
        val) {
57         int novo = create();
58
59         seg[novo] = seg[pos];
60         e[novo] = e[pos];
61         d[novo] = d[pos];
62
63         if (ini == fim) {
64             seg[novo] = val;
65             return novo;
66         }
67
68         int m = (ini + fim) >> 1;
69
70         if (id <= m) e[novo] = update(e[novo], ini, m
, id, val);
71         else d[novo] = update(d[novo], m + 1, fim, id
, val);
72
73         seg[novo] = f(seg[e[novo]], seg[d[novo]]);
74
75         return novo;
76     }
77
78     ftype query(int pos, int p, int q) {
79         return query(pos, 1, n, p, q);
80     }
81
82     int update(int pos, int id, int val) {
83         return update(pos, 1, n, id, val);
84     }
85 };
```

3.9 Dsu

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int MAX = 1e6+17;
6
7  struct DSU {
8      int n;
9      vector<int> link, sizes;
10
11     DSU(int n) {
12         this->n = n;
13         link.assign(n+1, 0);
14         sizes.assign(n+1, 1);
15
16         for (int i = 0; i <= n; i++)
17             link[i] = i;
18     }
19
20     int find(int x) {
21         while (x != link[x])
22             x = link[x];
23
24         return x;
25     }
26
27     bool same(int a, int b) {
28         return find(a) == find(b);
29     }
30
```

```cpp
    void unite(int a, int b) {
        a = find(a);
        b = find(b);

        if (a == b) return;

        if (sizes[a] < sizes[b])
            swap(a, b);

        sizes[a] += sizes[b];
        link[b] = a;
    }

    int size(int x) {
        return sizes[x];
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int cities, roads; cin >> cities >> roads;
    vector<int> final_roads;
    int ans = 0;
    DSU dsu = DSU(cities);
    for (int i = 0, a, b; i < roads; i++) {
        cin >> a >> b;
        dsu.unite(a, b);
    }

    for (int i = 2; i <= cities; i++) {
        if (!dsu.same(1, i)) {
            ans++;
            final_roads.push_back(i);
            dsu.unite(1,i);
        }
    }

    cout << ans << '\n';
    for (auto e : final_roads) {
        cout << "1 " << e << '\n';
    }

}
```

3.10   Ordered Set

```cpp
// Description:
// insert(k) - add element k to the ordered set
// erase(k) - remove element k from the ordered set
// erase(it) - remove element it points to from the
    ordered set
// order_of_key(k) - returns number of elements
    strictly smaller than k
// find_by_order(n) - return an iterator pointing to
    the k-th element in the ordered set (counting
    from zero).

// Problem:
// https://cses.fi/problemset/task/2169/

// Complexity:
// O(log n) for all operations

// How to use:
// ordered_set<int> os;
// cout << os.order_of_key(1) << '\n';
// cout << os.find_by_order(1) << '\n';

// Notes
// The ordered set only contains different elements
```

```cpp
// By using less_equal<T> instead of less<T> on using
    ordered_set declaration
// The ordered_set becomes an ordered_multiset
// So the set can contain elements that are equal

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T,null_type,less<T>,
    rb_tree_tag,tree_order_statistics_node_update>;
```

3.11   Priority Queue

```cpp
// Description:
// Keeps the largest (by default) element at the top
    of the queue

// Problem:
// https://cses.fi/problemset/task/1164/

// Complexity:
// O(log n) for push and pop
// O (1) for looking at the element at the top

// How to use:
// prioriy_queue<int> pq;
// pq.push(1);
// pq.top();
// pq.pop()

// Notes
// To use the priority queue keeping the smallest
    element at the top

priority_queue<int, vector<int>, greater<int>> pq;
```

4   Template

4.1   Template

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long
#define optimize std::ios::sync_with_stdio(false);
    cin.tie(NULL);
#define vi vector<int>
#define ll long long
#define pb push_back
#define mp make_pair
#define ff first
#define ss second
#define pii pair<int, int>
#define MOD 1000000007
#define sqr(x) ((x) * (x))
#define all(x) (x).begin(), (x).end()
#define FOR(i, j, n) for (int i = j; i < n; i++)
#define qle(i, n) (i == n ? "\n" : " ")
#define endl "\n"
const int oo = 1e9;
const int MAX = 1e6;

int32_t main(){ optimize;

    return 0;
}
```

4.2   Template Clean

```cpp
// Notes:
// Compile and execute
// g++ teste.cpp -o teste -std=c++17
// ./teste < teste.txt

// Print with precision
// cout << fixed << setprecision(12) << value << endl
    ;

// File as input and output
// freopen("input.txt", "r", stdin);
// freopen("output.txt", "w", stdout);

#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);


    return 0;
}
```

5 Graphs

5.1 Floyd Warshall

```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;

const int MAX  = 507;
const long long INF = 0x3f3f3f3f3f3f3f3fLL;

ll dist[MAX][MAX];
int n;

void floyd_warshall() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) dist[i][j] = 0;
            else if (!dist[i][j]) dist[i][j] = INF;
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                // trata o caso no qual o grafo tem
    arestas com peso negativo
                if (dist[i][k] < INF && dist[k][j] <
    INF){
                    dist[i][j] = min(dist[i][j], dist
    [i][k] + dist[k][j]);
                }
            }
        }
    }
}
```

5.2 Tree Diameter

```cpp
#include<bits/stdc++.h>

using namespace std;

const int MAX = 3e5+17;

```

```cpp
vector<int> adj[MAX];
bool visited[MAX];

int max_depth = 0, max_node = 1;

void dfs (int v, int depth) {
    visited[v] = true;

    if (depth > max_depth) {
        max_depth = depth;
        max_node = v;
    }

    for (auto u : adj[v]) {
        if (!visited[u]) dfs(u, depth + 1);
    }
}

int tree_diameter() {
    dfs(1, 0);
    max_depth = 0;
    for (int i = 0; i < MAX; i++) visited[i] = false;
    dfs(max_node, 0);
    return max_depth;
}
```

5.3 Cycle Path Recovery

```cpp
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v =
    parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
```

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

5.4 Bipartite

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

5.5 Find Cycle

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

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

5.6 Dinic

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

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

5.7 Bellman Ford

```
1 struct edge
2 {
3     int a, b, cost;
4 };
5
6 int n, m, v;
7 vector<edge> e;
8 const int INF = 1000000000;
9
10 void solve()
11 {
12     vector<int> d (n, INF);
13     d[v] = 0;
14     for (int i=0; i<n-1; ++i)
15         for (int j=0; j<m; ++j)
16             if (d[e[j].a] < INF)
17                 d[e[j].b] = min (d[e[j].b], d[e[j].a]
    + e[j].cost);
18 }
```

5.8 Dijkstra

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

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

5.9 Tarjan Bridge

```
1 // Description:
2 // Find a bridge in a connected unidirected graph
3 // A bridge is an edge so that if you remove that
    edge the graph is no longer connected
4
5 // Problem:
6 // https://cses.fi/problemset/task/2177/
7
8 // Complexity:
9 // O(V + E) where V is the number of vertices and E
    is the number of edges
10
11 int n;
12 vector<vector<int>> adj;
13
14 vector<bool> visited;
15 vector<int> tin, low;
16 int timer;
17
18 void dfs(int v, int p) {
```

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

5.10   Centroid Find

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

```
43        u--; v--;
44        adj[u].push_back(v);
45        adj[v].push_back(u);
46    }
47
48    get_subtree_size(0);
49    cout << get_centroid(0) + 1 << endl;
50 }
```

5.11   Small To Large

```
1  // Problem:
2  // https://codeforces.com/contest/600/problem/E
3
4  void process_colors(int curr, int parent) {
5
6    for (int n : adj[curr]) {
7      if (n != parent) {
8        process_colors(n, curr);
9
10         if (colors[curr].size() < colors[n].size
       ()) {
11             sum_num[curr] = sum_num[n];
12             vmax[curr] = vmax[n];
13         swap(colors[curr], colors[n]);
14       }
15
16       for (auto [item,vzs] : colors[n]) {
17             if(colors[curr][item]+vzs > vmax[curr
       ]){
18                 vmax[curr] = colors[curr][item] +
       vzs;
19                 sum_num[curr] = item;
20             }
21             else if(colors[curr][item]+vzs ==
       vmax[curr]){
22                 sum_num[curr] += item;
23             }
24
25             colors[curr][item] += vzs;
26       }
27     }
28   }
29
30 }
31
32
33 int32_t main() {
34
35   int n; cin >> n;
36
37   for (int i = 1; i <= n; i++) {
38     int a; cin >> a;
39     colors[i][a] = 1;
40       vmax[i] = 1;
41       sum_num[i] = a;
42   }
43
44   for (int i = 1; i < n; i++) {
45     int a, b; cin >> a >> b;
46
47     adj[a].push_back(b);
48     adj[b].push_back(a);
49   }
50
51   process_colors(1, 0);
52
53   for (int i = 1; i <= n; i++) {
54     cout << sum_num[i] << (i < n ? " " : "\n");
55   }
56
57     return 0;
58 }
```

```
59 }
60
```

5.12 Prim

```cpp
1  int n;
2  vector<vector<int>> adj; // adjacency matrix of graph
3  const int INF = 1000000000; // weight INF means there
       is no edge
4
5  struct Edge {
6      int w = INF, to = -1;
7  };
8
9  void prim() {
10     int total_weight = 0;
11     vector<bool> selected(n, false);
12     vector<Edge> min_e(n);
13     min_e[0].w = 0;
14
15     for (int i=0; i<n; ++i) {
16         int v = -1;
17         for (int j = 0; j < n; ++j) {
18             if (!selected[j] && (v == -1 || min_e[j].
   w < min_e[v].w))
19                 v = j;
20         }
21
22         if (min_e[v].w == INF) {
23             cout << "No MST!" << endl;
24             exit(0);
25         }
26
27         selected[v] = true;
28         total_weight += min_e[v].w;
29         if (min_e[v].to != -1)
30             cout << v << " " << min_e[v].to << endl;
31
32         for (int to = 0; to < n; ++to) {
33             if (adj[v][to] < min_e[to].w)
34                 min_e[to] = {adj[v][to], v};
35         }
36     }
37
38     cout << total_weight << endl;
39 }
```

5.13 Kruskall

```cpp
1  vector<int> parent, rank;
2
3  void make_set(int v) {
4      parent[v] = v;
5      rank[v] = 0;
6  }
7
8  int find_set(int v) {
9      if (v == parent[v])
10         return v;
11     return parent[v] = find_set(parent[v]);
12 }
13
14 void union_sets(int a, int b) {
15     a = find_set(a);
16     b = find_set(b);
17     if (a != b) {
18         if (rank[a] < rank[b])
19             swap(a, b);
20         parent[b] = a;
21         if (rank[a] == rank[b])
22             rank[a]++;
23     }
```

```cpp
24 }
25
26 struct Edge {
27     int u, v, weight;
28     bool operator<(Edge const& other) {
29         return weight < other.weight;
30     }
31 };
32
33 int n;
34 vector<Edge> edges;
35
36 int cost = 0;
37 vector<Edge> result;
38 parent.resize(n);
39 rank.resize(n);
40 for (int i = 0; i < n; i++)
41     make_set(i);
42
43 sort(edges.begin(), edges.end());
44
45 for (Edge e : edges) {
46     if (find_set(e.u) != find_set(e.v)) {
47         cost += e.weight;
48         result.push_back(e);
49         union_sets(e.u, e.v);
50     }
51 }
```

5.14 Lca

```cpp
1  // Description:
2  // Find the lowest common ancestor between two nodes
       in a tree
3
4  // Problem:
5  // https://cses.fi/problemset/task/1688/
6
7  // Complexity:
8  // O(log n)
9
10 // How to use:
11 // preprocess(1);
12 // lca(a, b);
13
14 // Notes
15 // To calculate the distance between two nodes use
       the following formula
16 // dist[a] + dist[b] - 2*dist[lca(a, b)]
17
18 const int MAX = 2e5+17;
19
20 const int BITS = 32;
21
22 vector<int> adj[MAX];
23 // vector<pair<int, int>> adj[MAX];
24 // int dist[MAX];
25
26 int timer;
27 vector<int> tin, tout;
28 vector<vector<int>> up;
29
30 void dfs(int v, int p)
31 {
32     tin[v] = ++timer;
33     up[v][0] = p;
34
35     for (int i = 1; i <= BITS; ++i) {
36         up[v][i] = up[up[v][i-1]][i-1];
37     }
38
39     for (auto u : adj[v]) {
40         if (u != p) {
```

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

5.15 Centroid Decomposition

```
1 int n;
2 vector<set<int>> adj;
3 vector<char> ans;
4
5 vector<bool> removed;
6
7 vector<int> subtree_size;
8
9 int dfs(int u, int p = 0) {
10   subtree_size[u] = 1;
11
12   for(int v : adj[u]) {
13     if(v != p && !removed[v]) {
14       subtree_size[u] += dfs(v, u);
15     }
16   }
17
18   return subtree_size[u];
19 }
20
21 int get_centroid(int u, int sz, int p = 0) {
22   for(int v : adj[u]) {
23     if(v != p && !removed[v]) {
24       if(subtree_size[v]*2 > sz) {
25         return get_centroid(v, sz, u);
26             }
27       }
28     }
29
30   return u;
31 }
```

```
32
33 char get_next(char c) {
34    if (c != 'Z') return c + 1;
35    return '$';
36 }
37
38 bool flag = true;
39
40 void solve(int node, char c) {
41   int center = get_centroid(node, dfs(node));
42    ans[center] = c;
43    removed[center] = true;
44
45    for (auto u : adj[center]) {
46        if (!removed[u]) {
47            char next = get_next(c);
48            if (next == '$') {
49                flag = false;
50                return;
51            }
52            solve(u, next);
53        }
54    }
55 }
56
57 int32_t main(){
58    ios::sync_with_stdio(false);
59    cin.tie(NULL);
60
61    cin >> n;
62    adj.resize(n + 1);
63    ans.resize(n + 1);
64    removed.resize(n + 1);
65    subtree_size.resize(n + 1);
66
67    for (int i = 1; i <= n - 1; i++) {
68        int u, v; cin >> u >> v;
69        adj[u].insert(v);
70        adj[v].insert(u);
71    }
72
73    solve(1, 'A');
74
75    if (!flag) cout << "Impossible!\n";
76    else {
77        for (int i = 1; i <= n; i++) {
78            cout << ans[i] << ' ';
79        }
80        cout << '\n';
81    }
82
83    return 0;
84 }
```

6 Strings

6.1 Kmp

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

6.2 Lcs

```
1 // Description:
2 // Finds the longest common subsquence 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] +
    1;
19       else
20         LCS_table[i][j] = max(LCS_table[i - 1][j],
    LCS_table[i][j - 1]);
21     }
22   }
23
24   int index = LCS_table[m][n];
25   char lcsAlgo[index + 1];
26   lcsAlgo[index] = '\0';
27
28   int i = m, j = n;
29   while (i > 0 && j > 0) {
30     if (s1[i - 1] == s2[j - 1]) {
31       lcsAlgo[index - 1] = s1[i - 1];
32       i--;
33       j--;
34       index--;
35     }
36
37     else if (LCS_table[i - 1][j] > LCS_table[i][j -
    1])
38       i--;
39     else
40       j--;
41   }
42
43   return lcsAlgo;
44 }
```

6.3 Generate All Permutations

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

6.4 Generate All Sequences Length K

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

6.5 Z-function

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

7 DP

7.1 Knapsack

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

7.2 Substr Palindrome

```
1 // êvoc deve informar se a substring de S formada
    pelos elementos entre os indices i e j
2 // é um palindromo ou ãno.
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 }
```

7.3   Edit Distance

```
1  // Description:
2  // Minimum number of operations required to transform
       a string into another
3  // Operations allowed: add character, remove
       character, replace character
4
5  // Parameters:
6  // str1 - string to be transformed into str2
7  // str2 - string that str1 will be transformed into
8  // m - size of str1
9  // n - size of str2
10
11 // Problem:
12 // https://cses.fi/problemset/task/1639
13
14 // Complexity:
15 // O(m x n)
16
17 // How to use:
18 // memset(dp, -1, sizeof(dp));
19 // string a, b;
20 // edit_distance(a, b, (int)a.size(), (int)b.size());
21
22 // Notes:
23 // Size of dp matriz is m x n
24
25 int dp[MAX][MAX];
26
27 int edit_distance(string &str1, string &str2, int m,
       int n) {
28    if (m == 0) return n;
29    if (n == 0) return m;
30
31    if (dp[m][n] != -1) return dp[m][n];
32
33    if (str1[m - 1] == str2[n - 1]) return dp[m][n] =
       edit_distance(str1, str2, m - 1, n - 1);
34    return dp[m][n] = 1 + min({edit_distance(str1,
       str2, m, n - 1), edit_distance(str1, str2, m - 1,
       n), edit_distance(str1, str2, m - 1, n - 1)});
35 }
```

7.4   Knapsack With Index

```
1  void knapsack(int W, int wt[], int val[], int n) {
2     int i, w;
3     int K[n + 1][W + 1];
4
5     for (i = 0; i <= n; i++) {
6         for (w = 0; w <= W; w++) {
7             if (i == 0 || w == 0)
8                 K[i][w] = 0;
9             else if (wt[i - 1] <= w)
10                K[i][w] = max(val[i - 1] +
                       K[i - 1][w - wt[i - 1]], K[i -
       1][w]);
12            else
```

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

7.5   Minimum Coin Change

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

7.6   Digits

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

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

7.7   Coins

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

7.8   Kadane

```
1 // achar uma subsequencia continua no array que a
       soma seja a maior possivel
2 // nesse caso vc precisa multiplicar exatamente 1
       elemento da subsequencia
3 // e achar a maior soma com isso
4
5 int n, x, arr[MAX], tab[MAX][2]; // tab[maior
       resposta no intervalo][foi multiplicado ou ãno]
6
7 int dp(int i, bool mult) {
8     if (i == n-1) {
9         if (!mult) return arr[n-1]*x;
10        return arr[n-1];
11    }
12    if (tab[i][mult] != -1) return tab[i][mult];
13
14    int res;
15
16    if (mult) {
17        res = max(arr[i], arr[i] + dp(i+1, 1));
18    }
19    else {
20        res = max({
21            arr[i]*x,
22            arr[i]*x + dp(i+1, 1),
23            arr[i] + dp(i+1, 0)
24        });
25    }
```

```
26    return tab[i][mult] = res;
27 }
28
30 int main() {
31
32    memset(tab, -1, sizeof(tab));
33
34    int ans = -oo;
35    for (int i = 0; i < n; i++) {
36        ans = max(ans, dp(i, 0));
37    }
38
39    return 0;
40 }
41
42
43
44 int ans = a[0], ans_l = 0, ans_r = 0;
45 int sum = 0, minus_pos = -1;
46
47 for (int r = 0; r < n; ++r) {
48     sum += a[r];
49     if (sum > ans) {
50         ans = sum;
51         ans_l = minus_pos + 1;
52         ans_r = r;
53     }
54     if (sum < 0) {
55         sum = 0;
56         minus_pos = r;
57     }
58 }
```

8   Math

8.1   Multiplicative Inverse

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

8.2   Divisors

```
1 vector<long long> all_divisors(long long n) {
2   vector<long long> ans;
3   for(long long a = 1; a*a <= n; a++){
4     if(n % a == 0) {
```

```
 5        long long b = n / a;
 6        ans.push_back(a);
 7        if(a != b) ans.push_back(b);
 8      }
 9    }
10    sort(ans.begin(), ans.end());
11    return ans;
12 }
```

8.3 Prime Factors

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

8.4 Binary To Decimal

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

8.5 Sieve Of Eratosthenes

```
 1 int n;
 2 vector<bool> is_prime(n+1, true);
 3 is_prime[0] = is_prime[1] = false;
 4 for (int i = 2; i <= n; i++) {
 5    if (is_prime[i] && (long long)i * i <= n) {
 6      for (int j = i * i; j <= n; j += i)
 7        is_prime[j] = false;
 8    }
 9 }
```

8.6 Check If Bit Is On

```
 1 // msb de 0 é undefined
```

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

8.7 Crt

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

8.8 Ceil

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

8.9 Matrix Exponentiation

```
 1 // Description:
 2 // Calculate the nth term of a linear recursion
 3
 4 // Example Fibonacci:
 5 // Given a linear recurrence, for example fibonacci
 6 // F(n) = n, x <= 1
 7 // F(n) = F(n - 1) + F(n - 2), x > 1
 8
 9 // The recurrence has two terms, so we can build a
      matrix 2 x 1 so that
10 // n + 1 = transition * n
11
12 // (2 x 1) = (2 x 2) * (2 x 1)
13 // F(n)       = a b * F(n - 1)
14 // F(n - 1)     c d   F(n - 2)
15
16 // Another Example:
17 // Given a grid 3 x n, you want to color it using 3
      distinct colors so that
18 // no adjacent place has the same color. In how many
      different ways can you do that?
19 // There are 6 ways for the first column to be
      colored using 3 distinct colors
20 // ans 6 ways using 2 equal colors and 1 distinct one
21
22 // Adding another column, there are:
23 // 3 ways to go from 2 equal to 2 equal
24 // 2 ways to go from 2 equal to 3 distinct
25 // 2 ways to go from 3 distinct to 2 equal
26 // 2 ways to go from 3 distinct to 3 distinct
27
28 // So we star with matrix 6 6 and multiply it by the
      transition 3 2 and get 18 12
29 //                      6 6
                  2 2         12 12
```

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

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

8.10   Linear Diophantine Equation

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

```cpp
54  void shift_solution(int & x, int & y, int a, int b,
        int cnt) {
55      x += cnt * b;
56      y -= cnt * a;
57  }
58
59  // return number of solutions in the interval
60  int find_all_solutions(int a, int b, int c, int minx,
         int maxx, int miny, int maxy) {
61      int x, y, g;
62      if (!find_any_solution(a, b, c, x, y, g))
63          return 0;
64      a /= g;
65      b /= g;
66
67      int sign_a = a > 0 ? +1 : -1;
68      int sign_b = b > 0 ? +1 : -1;
69
70      shift_solution(x, y, a, b, (minx - x) / b);
71      if (x < minx)
72          shift_solution(x, y, a, b, sign_b);
73      if (x > maxx)
74          return 0;
75      int lx1 = x;
76
77      shift_solution(x, y, a, b, (maxx - x) / b);
78      if (x > maxx)
79          shift_solution(x, y, a, b, -sign_b);
80      int rx1 = x;
81
82      shift_solution(x, y, a, b, -(miny - y) / a);
83      if (y < miny)
84          shift_solution(x, y, a, b, -sign_a);
85      if (y > maxy)
86          return 0;
87      int lx2 = x;
88
89      shift_solution(x, y, a, b, -(maxy - y) / a);
90      if (y > maxy)
91          shift_solution(x, y, a, b, sign_a);
92      int rx2 = x;
93
94      if (lx2 > rx2)
95          swap(lx2, rx2);
96      int lx = max(lx1, lx2);
97      int rx = min(rx1, rx2);
98
99      if (lx > rx)
100         return 0;
101     return (rx - lx) / abs(b) + 1;
102 }
```

8.11   Fast Exponentiation

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

9   Algorithms

9.1   Lis

```cpp
1  int lis(vector<int> const& a) {
```

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

9.2   Ternary Search

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

9.3   Binary Search First True

```cpp
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 }
```

9.4   Delta-encoding

```cpp
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;
```

```cpp
    for(int i=0; i < n; i++){
        curr += delta[i];
        v[i] = curr;
    }

    for(int i=0; i< n; i++){
        cout << v[i] << ' ';
    }
    cout << '\n';

    return 0;
}
```

9.5 Binary Search Last True

```cpp
int last_true(int lo, int hi, function<bool(int)> f)
    {
    lo--;
    while (lo < hi) {
        int mid = lo + (hi - lo + 1) / 2;
        if (f(mid)) {
            lo = mid;
        } else {
            hi = mid - 1;
        }
    }
    return lo;
}
```