



# Notebook - Maratona de Programação

Lenhadoras de Segtree

## Contents

### 1 Misc

1.1	Builtin Overflow	2
1.2	Split	2
1.3	Lower Upper	2
1.4	Eraseunique	2
1.5	Int128	2

### 2 Data Structures

2.1	Ordered Set	2
2.2	Sparse Table	3
2.3	Dsu	3
2.4	Mergesort Tree Ordered Set	4
2.5	Two Sets	5
2.6	Sparse Table2d	5
2.7	Priority Queue	6
2.8	Psum2d	6
2.9	Mergesort Tree Vector	6
2.10	Segment With Maximum Sum	7
2.11	Minimum And Amount	8
2.12	Dynamic Implicit Sparse	9
2.13	Range Query Point Update	10
2.14	Segtree2d	11
2.15	Lazy Addition To Segment	12
2.16	Lazy Assignment To Segment	13
2.17	Lazy Dynamic Implicit Sparse	14
2.18	Persistent	15

### 3 Strings

3.1	Lcs	15
3.2	Z-function	16
3.3	Hash2	16

3.4	Trie	16
3.5	Generate All Permutations	17
3.6	Kmp	17
3.7	Hash	17
3.8	Generate All Sequences Length K	17
3.9	Suffix Array	17

### 4 Algorithms

4.1	Lcs	19
4.2	Biggest K	19
4.3	Lis	20
4.4	Binary Search First True	20
4.5	Subsets	20
4.6	Ternary Search	20
4.7	Binary Search Last True	20
4.8	Delta-encoding	20
4.9	Binary Search Real	21

### 5 Template

5.1	Template	21
5.2	Template Clean	21

### 6 Math

6.1	Crt	21
6.2	Check If Bit Is On	21
6.3	To Decimal	22
6.4	Multiplicative Inverse	22
6.5	Fft	22
6.6	Function Root	22
6.7	Set Operations	22
6.8	Subsets	23
6.9	Pascalsrule Stifel	23
6.10	Sieve Of Eratosthenes	23

6.11	Linear Diophantine Equation . . . . .	23	9	<b>Geometry</b>	<b>48</b>
6.12	Mobius . . . . .	24	9.1	Shoelace Boundary . . . . .	48
6.13	Representation Arbitrary Base . . . . .	24	9.2	Mindistpair . . . . .	48
6.14	Horner Algorithm . . . . .	24	9.3	2d . . . . .	48
6.15	Prime Factors . . . . .	24	9.4	Inside Polygon . . . . .	51
6.16	Binary To Decimal . . . . .	24	9.5	Convexhull . . . . .	51
6.17	Matrix Exponentiation . . . . .	25	9.6	Delaunay . . . . .	52
6.18	Fast Exponentiation . . . . .	25	9.7	Closest Pair Points . . . . .	54
6.19	Divisors . . . . .	26			
6.20	Ntt . . . . .	26			
6.21	Phi . . . . .	27			
6.22	Division Trick . . . . .	27			
6.23	Ceil . . . . .	28			
<b>7</b>	<b>DP</b>	<b>28</b>			
7.1	Knapsack . . . . .	28			
7.2	Knapsack With Index . . . . .	28			
7.3	Substr Palindrome . . . . .	28			
7.4	Edit Distance . . . . .	28			
7.5	Coins . . . . .	29			
7.6	Digits . . . . .	29			
7.7	Minimum Coin Change . . . . .	29			
7.8	Kadane . . . . .	29			
7.9	Divide And Conquer . . . . .	30			
7.10	Knuth . . . . .	30			
7.11	Cht . . . . .	31			
<b>8</b>	<b>Graphs</b>	<b>31</b>			
8.1	Dijkstra . . . . .	31			
8.2	Bipartite . . . . .	32			
8.3	Eulerian Undirected . . . . .	32			
8.4	Kruskall . . . . .	33			
8.5	Negative Cycle . . . . .	34			
8.6	Floyd Warshall . . . . .	34			
8.7	Centroid Find . . . . .	34			
8.8	Eulerian Directed . . . . .	35			
8.9	Lca . . . . .	36			
8.10	Ford Fulkerson Edmonds Karp . . . . .	36			
8.11	Kuhn . . . . .	37			
8.12	Cycle Path Recovery . . . . .	37			
8.13	Hld Vertex . . . . .	38			
8.14	Small To Large . . . . .	39			
8.15	Centroid Decomposition . . . . .	40			
8.16	Min Cost Max Flow . . . . .	40			
8.17	Hungarian . . . . .	41			
8.18	2sat . . . . .	41			
8.19	Tarjan Bridge . . . . .	43			
8.20	Find Cycle . . . . .	43			
8.21	Prim . . . . .	44			
8.22	Blossom . . . . .	44			
8.23	Dinic . . . . .	44			
8.24	Bellman Ford . . . . .	46			
8.25	Hld Edge . . . . .	47			
8.26	Tree Diameter . . . . .	48			

# 1 Misc

## 1.1 Builtin Overflow

```
1 // Returns true if the operation results in overflow.
2
3 bool __builtin_add_overflow (type1 a, type2 b, type3
    *res)
4 bool __builtin_sadd_overflow (int a, int b, int *res)
5 bool __builtin_saddll_overflow (long int a, long int b
    , long int *res)
6 bool __builtin_saddll_overflow (long long int a, long
    long int b, long long int *res)
7 bool __builtin_uadd_overflow (unsigned int a,
    unsigned int b, unsigned int *res)
8 bool __builtin_uaddl_overflow (unsigned long int a,
    unsigned long int b, unsigned long int *res)
9 bool __builtin_uaddll_overflow (unsigned long long
    int a, unsigned long long int b, unsigned long
    long int *res)
10
11 bool __builtin_sub_overflow (type1 a, type2 b, type3
    *res)
12 bool __builtin_ssub_overflow (int a, int b, int *res)
13 bool __builtin_ssubl_overflow (long int a, long int b
    , long int *res)
14 bool __builtin_ssubll_overflow (long long int a, long
    long int b, long long int *res)
15 bool __builtin_usub_overflow (unsigned int a,
    unsigned int b, unsigned int *res)
16 bool __builtin_usubl_overflow (unsigned long int a,
    unsigned long int b, unsigned long int *res)
17 bool __builtin_usubll_overflow (unsigned long long
    int a, unsigned long long int b, unsigned long
    long int *res)
18
19 bool __builtin_mul_overflow (type1 a, type2 b, type3
    *res)
20 bool __builtin_smul_overflow (int a, int b, int *res)
21 bool __builtin_smull_overflow (long int a, long int b
    , long int *res)
22 bool __builtin_smulll_overflow (long long int a, long
    long int b, long long int *res)
23 bool __builtin_umul_overflow (unsigned int a,
    unsigned int b, unsigned int *res)
24 bool __builtin_umull_overflow (unsigned long int a,
    unsigned long int b, unsigned long int *res)
25 bool __builtin_umulll_overflow (unsigned long long
    int a, unsigned long long int b, unsigned long
    long int *res)
26
27 bool __builtin_add_overflow_p (type1 a, type2 b,
    type3 c)
28 bool __builtin_sub_overflow_p (type1 a, type2 b,
    type3 c)
29 bool __builtin_mul_overflow_p (type1 a, type2 b,
    type3 c)
```

## 1.2 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
18 if(palTemp.size() > 0)
19     ans.push_back(palTemp);
20
21 return ans;
22 }
```

## 1.3 Lower Upper

```
1 cout << (char)tolower('A') << '\n';
```

## 1.4 Eraseunique

```
1 sort(v.begin(), v.end());
2 v.erase(unique(v.begin(), v.end()), v.end());
```

## 1.5 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 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 }

```

## 2.2 Sparse Table

```

1 // Description:
2 // Data structure to query for minimum and maximum
3
4 // Problem:
5 // https://cses.fi/problemset/task/1647/
6
7 // Complexity:
8 // Build  $O(n \log n)$ 
9 // Query  $O(1)$ 
10
11 #include <bits/stdc++.h>
12
13 using namespace std;
14
15 const int MAX = 2e5+17;
16 const int INF = 1e9+17;
17
18 struct SparseTable {
19     int n;
20     vector<int> arr;
21     vector<vector<int>> st;
22     vector<int> log_2;
23
24     SparseTable(vector<int>& arr, int& n) : arr(arr), n
        (n) {
25         build();
26     }
27
28     void build() {
29         log_2.resize(MAX + 1);
30
31         log_2[1] = 0;
32         for (int i = 2; i <= MAX; i++) {
33             log_2[i] = log_2[i/2] + 1;
34         }
35
36         int K = log_2[n + 1];
37
38         st.resize(MAX, vector<int>(K + 1));
39
40         for (int i = 0; i < MAX; i++) {
41             for (int j = 0; j < K + 1; j++) {
42                 st[i][j] = INF;
43             }
44         }
45
46         for (int i = 0; i < n; i++) {
47             st[i][0] = arr[i];
48         }
49
50         for (int j = 1; j <= K; j++) {
51             for (int i = 0; i + (1 << j) < MAX; i++) {

```

```

                st[i][j] = min(st[i][j-1], st[i + (1 << (j -
                    1))] [j - 1]);
            }
        }
    }
}

int query(int l, int r) {
    int j = log_2[r - l + 1];
    return min(st[l][j], st[r - (1 << j) + 1][j]);
}
};

```

## 2.3 Dsu

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX = 1e6+17;
6
7 struct DSU {
8     int n;
9     vector<int> link, sizes;
10
11     DSU(int n) {
12         this->n = n;
13         link.assign(n+1, 0);
14         sizes.assign(n+1, 1);
15
16         for (int i = 0; i <= n; i++)
17             link[i] = i;
18     }
19
20     int find(int x) {
21         while (x != link[x])
22             x = link[x];
23
24         return x;
25     }
26
27     bool same(int a, int b) {
28         return find(a) == find(b);
29     }
30
31     void unite(int a, int b) {
32         a = find(a);
33         b = find(b);
34
35         if (a == b) return;
36
37         if (sizes[a] < sizes[b])
38             swap(a, b);
39
40         sizes[a] += sizes[b];
41         link[b] = a;
42     }
43
44     int size(int x) {
45         return sizes[x];
46     }
47 };
48
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(NULL);
52
53     int cities, roads; cin >> cities >> roads;
54     vector<int> final_roads;
55     int ans = 0;
56     DSU dsu = DSU(cities);
57     for (int i = 0, a, b; i < roads; i++) {
58         cin >> a >> b;
59         dsu.unite(a, b);

```

```

60     }
61
62     for (int i = 2; i <= cities; i++) {
63         if (!dsu.same(1, i)) {
64             ans++;
65             final_roads.push_back(i);
66             dsu.unite(1, i);
67         }
68     }
69
70     cout << ans << '\n';
71     for (auto e : final_roads) {
72         cout << "1 " << e << '\n';
73     }
74
75 }

```

## 2.4 Mergesort Tree Ordered Set

```

1 // Description:
2 // In each node, the tree keeps a sorted list of
3 // elements in that range.
4 // It can be used to find how many elements are
5 // greater than x in a given range.
6 // It can also be used to find the position of an
7 // element if the list was sorted.
8 // query(i, j, k) - how many elements greater than k
9 // are in the range (i, j)
10 // update(i, val) - changes the value of the element
11 // on index i to val
12
13 // Problem:
14 // https://www.beecrowd.com.br/judge/pt/problems/view/3097
15
16 // Complexity:
17 // 0(n log ^ 2 ~ 2 n) for build
18 // 0(log ^ 2 n) for query
19
20 #include <ext/pb_ds/assoc_container.hpp>
21 #include <ext/pb_ds/tree_policy.hpp>
22
23 using namespace __gnu_pbds;
24 template <typename T>
25 using ordered_set = tree<T, null_type, less_equal<T>,
26 rb_tree_tag, tree_order_statistics_node_update>;
27
28 struct MergeSortTree {
29     vector<ordered_set<int>> tree;
30     vector<int> v;
31     int n;
32
33     MergeSortTree(int n, vector<int>& v) : n(n), v(v) {
34         int sz = 1;
35         while (sz < n) sz *= 2;
36
37         tree.resize(2 * sz);
38
39         build(0, 0, n - 1, v);
40     }
41
42     void Erase(ordered_set<int>& a, int x) {
43         int r = a.order_of_key(x);
44         auto it = a.find_by_order(r);
45         a.erase(it);
46     }
47
48     ordered_set<int> merge(ordered_set<int>& a,
49 ordered_set<int>& b) {
50     ordered_set<int> res;
51
52     for (auto e : a) res.insert(e);
53     for (auto e : b) res.insert(e);
54 }

```

```

47     return res;
48 }
49
50 void build(int pos, int ini, int fim, vector<int>&
v) {
51     if (ini == fim) {
52         if (ini < (int)v.size()) {
53             tree[pos].insert(v[ini]);
54         }
55         return;
56     }
57
58     int mid = ini + (fim - ini) / 2;
59
60     build(2 * pos + 1, ini, mid, v);
61     build(2 * pos + 2, mid + 1, fim, v);
62
63     tree[pos] = merge(tree[2 * pos + 1], tree[2 * pos
+ 2]);
64 }
65
66 // how many elements greater than val in vector v
67 int search(ordered_set<int>& v, int val) {
68     return (int)v.size() - v.order_of_key(val + 1);
69 }
70
71 // how many elements greater than val in the range
72 (p, q)
73 int query(int pos, int ini, int fim, int p, int q,
int val) {
74     if (fim < p || ini > q) {
75         return 0;
76     }
77
78     if (ini >= p && fim <= q) {
79         return search(tree[pos], val);
80     }
81
82     int mid = ini + (fim - ini) / 2;
83     return query(2 * pos + 1, ini, mid, p, q, val) +
84 query(2 * pos + 2, mid + 1, fim, p, q, val);
85 }
86
87 void update(int pos, int ini, int fim, int id, int
val) {
88     if (ini == id && fim == id) {
89         if (!tree[pos].empty()) Erase(tree[pos], v[id]);
90     };
91     tree[pos].insert(val);
92     return;
93 }
94
95 if (fim < id || ini > id) {
96     return;
97 }
98
99 int mid = ini + (fim - ini) / 2;
100 update(2 * pos + 1, ini, mid, id, val);
101 update(2 * pos + 2, mid + 1, fim, id, val);
102
103 if (!tree[pos].empty()) Erase(tree[pos], v[id]);
104 tree[pos].insert(val);
105 }
106
107 int query(int p, int q, int val) {
108     return query(0, 0, n - 1, p, q, val);
109 }
110
111 void update(int id, int val) {
112     update(0, 0, n - 1, id, val);
113     v[id] = val;
114 }

```

```
113 };
```

## 2.5 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 };
```

## 2.6 Sparse Table2d

```
1 // Description
2 // Minimum queries in a 2D grid
3
4 // Problem:
5 // https://codeforces.com/group/YgJnumGtHD/contest
  /103794/problem/D
6
7 // Complexity:
8 // Build  $O(N * M * \log(N) * \log(M))$ 
9 // Query  $O(1)$ 
10 // Memory CComplexity:  $O(N * M * \log(N) * \log(M))$ 
11
12 const int MAX = 410;
13
14 struct SparseTable2D {
15     vector<vector<int>>> matrix;
16     vector<vector<vector<vector<int>>>> table;
17     int n, m;
18
19     SparseTable2D(vector<vector<int>>& matrix, int n,
        int m) : matrix(matrix), n(n), m(m) {
20         table.resize(MAX, vector<vector<vector<int>>>(MAX
            , vector<vector<int>>>(log2(MAX) + 1, vector<int>(
                log2(MAX) + 1))));
21         build();
22     }
23
24     int f(int a, int b) {
25         return max(a, b);
26     }
27
28     void build() {
29         for (int i = 0; i < n; i++) {
30             for (int j = 0; j < m; j++) {
31                 table[i][j][0][0] = matrix[i][j];
32             }
33         }
34
35         for (int k = 1; k <= (int)(log2(n)); k++) {
36             for (int i = 0; i + (1 << k) - 1 < n; i++) {
37                 for (int j = 0; j + (1 << k) - 1 < m; j++) {
```

```

38         table[i][j][k][0] = f(
39             table[i][j][k - 1][0],
40             table[i + (1 << (k - 1))][j][k - 1][0]);
41     }
42 }
43 }
44
45 for (int k = 1; k <= (int)(log2(m)); k++) {
46     for (int i = 0; i < n; i++) {
47         for (int j = 0; j + (1 << k) - 1 < m; j++) {
48             table[i][j][0][k] = f(
49                 table[i][j][0][k - 1],
50                 table[i][j + (1 << (k - 1))][0][k - 1]);
51         }
52     }
53 }
54
55 for (int k = 1; k <= (int)(log2(n)); k++) {
56     for (int l = 1; l <= (int)(log2(m)); l++) {
57         for (int i = 0; i + (1 << k) - 1 < n; i++) {
58             for (int j = 0; j + (1 << l) - 1 < m; j++) {
59
60                 table[i][j][k][l] = f(
61                     f(
62                         table[i][j][k - 1][l - 1],
63                         table[i + (1 << (k - 1))][j][k - 1][l - 1]
64                     ),
65                     f(
66                         table[i][j + (1 << (l - 1))][k - 1][l - 1],
67                         table[i + (1 << (k - 1))][j + (1 << (l - 1))][k - 1][l - 1]
68                     )
69                 );
70             }
71         }
72     }
73
74     int query(int x1, int y1, int x2, int y2) {
75         int k = log2(x2 - x1 + 1);
76         int l = log2(y2 - y1 + 1);
77
78         return f(
79             f(
80                 table[x1][y1][k][l],
81                 table[x2 - (1 << k) + 1][y1][k][l]
82             ),
83             f(
84                 table[x1][y2 - (1 << l) + 1][k][l],
85                 table[x2 - (1 << k) + 1][y2 - (1 << l) + 1][k][l]
86             )
87         );
88     }
89 };

```

## 2.7 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;

```

## 2.8 Psum2d

```

1 // Description:
2 // Queries the sum of a rectangle that goes from grid
  [from_row][from_col] to grid[to_row][to_col]
3
4 // Problem:
5 // https://cses.fi/problemset/task/1652/
6
7 // Complexity:
8 // O(n) build
9 // O(1) query
10
11 for (int i = 1; i <= n; i++) {
12     for (int j = 1; j <= n; j++) {
13         psum[i][j] = grid[i][j] + psum[i - 1][j] + psum[i][j - 1] - psum[i - 1][j - 1];
14     }
15 }
16
17 while (q--) {
18     int from_row, to_row, from_col, to_col;
19     cin >> from_row >> from_col >> to_row >> to_col;
20     cout << psum[to_row][to_col] - psum[from_row - 1][to_col] - psum[to_row][from_col - 1] + psum[from_row - 1][from_col - 1] << '\n';
21 }
22 }

```

## 2.9 Mergesort Tree Vector

```

1 // Description:
2 // In each node, the tree keeps a sorted list of
  elements in that range.
3 // It can be used to find how many elements are
  greater than x in a given range.
4 // It can also be used to find the position of an
  element if the list was sorted.
5 // query(i, j, k) - how many elements greater than k
  are in the range (i, j)
6
7 // Problem:
8 // https://www.spoj.com/problems/KQUERY
9
10 // Complexity:
11 // O(n log n) for build
12 // O(log2 n) for query
13
14 struct MergeSortTree {
15     vector<vector<int>> tree;
16     int n;
17
18     MergeSortTree(int n, vector<int>& v) : n(n) {
19         int sz = 1;
20         while (sz < n) sz *= 2;
21
22         tree.assign(2 * sz, vector<int>());
23         build(0, 0, n - 1, v);
24     }
25
26     vector<int> merge(vector<int>& a, vector<int>& b) {
27         vector<int> res((int)a.size() + (int)b.size());

```

```

28     int it = 0, jt = 0, curr = 0;
29
30     while (it < (int)a.size() && jt < (int)b.size())
31     {
32         if (a[it] <= b[jt]) {
33             res[curr++] = a[it++];
34         } else {
35             res[curr++] = b[jt++];
36         }
37     }
38
39     while (it < (int)a.size()) {
40         res[curr++] = a[it++];
41     }
42
43     while (jt < (int)b.size()) {
44         res[curr++] = b[jt++];
45     }
46
47     return res;
48 }
49
50 void build(int pos, int ini, int fim, vector<int>&
51 v) {
52     if (ini == fim) {
53         if (ini < (int)v.size()) {
54             tree[pos].pb(v[ini]);
55         }
56         return;
57     }
58
59     int mid = ini + (fim - ini) / 2;
60
61     build(2 * pos + 1, ini, mid, v);
62     build(2 * pos + 2, mid + 1, fim, v);
63
64     tree[pos] = merge(tree[2 * pos + 1], tree[2 * pos
65 + 2]);
66 }
67
68 // how many elements greater than val in vector v
69 int search(vector<int>& v, int val) {
70     auto it = upper_bound(v.begin(), v.end(), val);
71     if (it == v.end()) return 0;
72     return (int)v.size() - (it - v.begin());
73 }
74
75 // how many elements greater than val in the range
76 (p, q)
77 int query(int pos, int ini, int fim, int p, int q,
78 int val) {
79     if (fim < p || ini > q) {
80         return 0;
81     }
82
83     if (ini >= p && fim <= q) {
84         return search(tree[pos], val);
85     }
86
87     int mid = ini + (fim - ini) / 2;
88     return query(2 * pos + 1, ini, mid, p, q, val) +
89     query(2 * pos + 2, mid + 1, fim, p, q, val);
90 }
91
92 int query(int p, int q, int val) {
93     return query(0, 0, n - 1, p, q, val);
94 }
95 };

```

## 2.10 Segment With Maximum Sum

1 // Description:

```

2 // Query - get sum of segment that is maximum among
3 // all segments
4 // E.g
5 // Array: 5 -4 4 3 -5
6 // Maximum segment sum: 8 because 5 + (-4) + 4 + 3 =
7 // 8
8 // Update - update element at position id to a value
9 // val
10
11 // Problem:
12 // https://codeforces.com/edu/course/2/lesson/4/2/
13 // practice/contest/273278/problem/A
14
15 // Complexity:
16 // O(log n) for both query and update
17
18 // How to use:
19 // Segtree seg = Segtree(n);
20 // seg.build(v);
21
22 // Notes
23 // The maximum segment sum can be a negative number
24 // In that case, taking zero elements is the best
25 // choice
26 // So we need to take the maximum between 0 and the
27 // query
28 // max(0LL, seg.query(0, n).max_seg)
29
30 using ll = long long;
31
32 typedef ll ftype_node;
33
34 struct Node {
35     ftype_node max_seg;
36     ftype_node pref;
37     ftype_node suf;
38     ftype_node sum;
39
40     Node(ftype_node max_seg, ftype_node pref,
41         ftype_node suf, ftype_node sum) : max_seg(max_seg
42 ), pref(pref), suf(suf), sum(sum) {};
43 };
44
45 typedef Node ftype;
46
47 struct Segtree {
48     vector<ftype> seg;
49     int n;
50     const ftype NEUTRAL = Node(0, 0, 0, 0);
51
52     Segtree(int n) {
53         int sz = 1;
54         // potencia de dois mais proxima
55         while (sz < n) sz *= 2;
56         this->n = sz;
57
58         // numero de nos da seg
59         seg.assign(2*sz, NEUTRAL);
60     }
61
62     ftype f(ftype a, ftype b) {
63         ftype_node max_seg = max({a.max_seg, b.
64 max_seg, a.suf + b.pref});
65         ftype_node pref = max(a.pref, a.sum + b.pref)
66 ;
67         ftype_node suf = max(b.suf, b.sum + a.suf);
68         ftype_node sum = a.sum + b.sum;
69
70         return Node(max_seg, pref, suf, sum);
71     }
72
73     ftype query(int pos, int ini, int fim, int p, int
74 q) {

```



```

64     if (ini >= p && fim <= q) {
65         return seg[pos];
66     }
67
68     if (q < ini || p > fim) {
69         return NEUTRAL;
70     }
71
72     int e = 2*pos + 1;
73     int d = 2*pos + 2;
74     int m = ini + (fim - ini) / 2;
75
76     return f(query(e, ini, m, p, q), query(d, m +
77 1, fim, p, q));
78
79 void update(int pos, int ini, int fim, int id,
80 int val) {
81     if (ini > id || fim < id) {
82         return;
83     }
84
85     if (ini == id && fim == id) {
86         seg[pos] = Node(val, val, val, val);
87
88         return;
89     }
90
91     int e = 2*pos + 1;
92     int d = 2*pos + 2;
93     int m = ini + (fim - ini) / 2;
94
95     update(e, ini, m, id, val);
96     update(d, m + 1, fim, id, val);
97
98     seg[pos] = f(seg[e], seg[d]);
99
100 void build(int pos, int ini, int fim, vector<int>
101 &v) {
102     if (ini == fim) {
103         // se a posição existir no array original
104         // seg tamanho potencia de dois
105         if (ini < (int)v.size()) {
106             seg[pos] = Node(v[ini], v[ini], v[ini]
107 ], v[ini]);
108         }
109         return;
110     }
111
112     int e = 2*pos + 1;
113     int d = 2*pos + 2;
114     int m = ini + (fim - ini) / 2;
115
116     build(e, ini, m, v);
117     build(d, m + 1, fim, v);
118
119     seg[pos] = f(seg[e], seg[d]);
120
121 }
122
123 ftype query(int p, int q) {
124     return query(0, 0, n - 1, p, q);
125 }
126
127 void update(int id, int val) {
128     update(0, 0, n - 1, id, val);
129 }
130
131 void build(vector<int> &v) {
132     build(0, 0, n - 1, v);
133 }
134
135 void debug() {

```

```

133     for (auto e : seg) {
134         cout << e.max_seg << ' ' << e.pref << ' '
135         << e.suf << ' ' << e.sum << '\n';
136     }
137     cout << '\n';
138 };

```

## 2.11 Minimum And Amount

```

1 // Description:
2 // Query - get minimum element in a range (l, r)
3 // inclusive
4 // and also the number of times it appears in that
5 // range
6 // Update - update element at position id to a value
7 // val
8
9 // Problem:
10 // https://codeforces.com/edu/course/2/lesson/4/1/
11 // practice/contest/273169/problem/C
12
13 // Complexity:
14 // O(log n) for both query and update
15
16 // How to use:
17 // Segtree seg = Segtree(n);
18 // seg.build(v);
19
20 #define pii pair<int, int>
21 #define mp make_pair
22 #define ff first
23 #define ss second
24
25 const int INF = 1e9+17;
26
27 typedef pii ftype;
28
29 struct Segtree {
30     vector<ftype> seg;
31     int n;
32     const ftype NEUTRAL = mp(INF, 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         if (a.ff < b.ff) return a;
44         if (b.ff < a.ff) return b;
45
46         return mp(a.ff, a.ss + b.ss);
47     }
48
49     ftype query(int pos, int ini, int fim, int p, int
50 q) {
51     if (ini >= p && fim <= q) {
52         return seg[pos];
53     }
54
55     if (q < ini || p > fim) {
56         return NEUTRAL;
57     }
58
59     int e = 2*pos + 1;
60     int d = 2*pos + 2;
61     int m = ini + (fim - ini) / 2;

```

```

58     return f(query(e, ini, m, p, q), query(d, m + 5 // and the queries need to be answered online so we
1, fim, p, q)); // can't sort the nodes and compress them
59 } // we create nodes only when they are needed so there
60 // 'll be (Q*log(MAX)) nodes
61 void update(int pos, int ini, int fim, int id, 7 // where Q is the number of queries and MAX is the
int val) { // maximum index a node can assume
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> 25 const int MAX = 1e9+17;
&v) { // 26 struct Segtree {
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 };

```

## 2.12 Dynamic Implicit Sparse

```

1 // Description:
2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
   stored in an array

```

```

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

```

## 2.13 Range Query Point Update

```

1 // Description:
2 // Indexed at zero
3 // Query - get sum of elements from range (l, r)
4 // inclusive
5 // Update - update element at position id to a value
6 // val
7 // Problem:
8 // https://codeforces.com/edu/course/2/lesson/4/1/
9 // practice/contest/273169/problem/B
10 // Complexity:
11 // O(log n) for both query and update
12 // How to use:
13 // Segtree seg = Segtree(n);
14 // seg.build(v);
15 // Notes
16 // Change neutral element and f function to perform a
17 // different operation
18 // If you want to change the operations to point
19 // query and range update
20 // Use the same segtree, but perform the following
21 // operations
22 // Query - seg.query(0, id);
23 // Update - seg.update(l, v); seg.update(r + 1, -v);
24
25 typedef long long ftype;
26
27 struct Segtree {
28     vector<ftype> seg;
29     int n;
30     const ftype NEUTRAL = 0;
31
32     Segtree(int n) {
33         int sz = 1;
34         while (sz < n) sz *= 2;
35         this->n = sz;
36
37         seg.assign(2*sz, NEUTRAL);
38     }
39
40     ftype f(ftype a, ftype b) {
41         return a + b;
42     }
43
44     ftype query(int pos, int ini, int fim, int p, int
45         q) {
46         if (ini >= p && fim <= q) {

```

```

47             return seg[pos];
48         }
49
50         if (q < ini || p > fim) {
51             return NEUTRAL;
52         }
53
54         int e = 2*pos + 1;
55         int d = 2*pos + 2;
56         int m = ini + (fim - ini) / 2;
57
58         return f(query(e, ini, m, p, q), query(d, m +
59             1, fim, p, q));
60     }
61
62     void update(int pos, int ini, int fim, int id,
63         int val) {
64         if (ini > id || fim < id) {
65             return;
66         }
67
68         if (ini == id && fim == id) {
69             seg[pos] = val;
70
71             return;
72         }
73
74         int e = 2*pos + 1;
75         int d = 2*pos + 2;
76         int m = ini + (fim - ini) / 2;
77
78         update(e, ini, m, id, val);
79         update(d, m + 1, fim, id, val);
80
81         seg[pos] = f(seg[e], seg[d]);
82     }
83
84     void build(int pos, int ini, int fim, vector<int>
85         &v) {
86         if (ini == fim) {
87             if (ini < (int)v.size()) {
88                 seg[pos] = v[ini];
89             }
90             return;
91         }
92
93         int e = 2*pos + 1;
94         int d = 2*pos + 2;
95         int m = ini + (fim - ini) / 2;
96
97         build(e, ini, m, v);
98         build(d, m + 1, fim, v);
99
100         seg[pos] = f(seg[e], seg[d]);
101     }
102
103     ftype query(int p, int q) {
104         return query(0, 0, n - 1, p, q);
105     }
106
107     void update(int id, int val) {
108         update(0, 0, n - 1, id, val);
109     }
110
111     void build(vector<int> &v) {
112         build(0, 0, n - 1, v);
113     }
114
115     void debug() {
116         for (auto e : seg) {
117             cout << e << ' ';
118         }
119         cout << '\n';

```

```

115     }
116 };

2.14 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(i, j) - update the value of grid[i][j]
6 // query(i1, j1, i2, j2) - return the sum of values
7 // inside the rectangle
8 // defined by grid[i1][j1] and grid[i2][j2] inclusive
9 // Problem:
10 // https://cses.fi/problemset/task/1739/
11 // Complexity:
12 // Time complexity:
13 // O(log N * log M) for both query and update
14 // O(N * M) for build
15 // Memory complexity:
16 // 4 * M * N
17
18 // How to use:
19 // Segtree2D seg = Segtree2D(n, m);
20 // vector<vector<int>> v(n, vector<int>(m));
21 // seg.build(v);
22
23 struct Segtree2D {
24     const int MAXN = 1025;
25     const int NEUTRAL = 0;
26     int N, M;
27
28     vector<vector<int>> seg;
29
30     Segtree2D(int N, int M) {
31         this->N = N;
32         this->M = M;
33         seg.assign(4*MAXN, vector<int>(4*MAXN,
34 NEUTRAL));
35     }
36
37     int f(int a, int b) {
38         return max(a, b);
39     }
40
41     void buildY(int noX, int lX, int rX, int noY, int
42 lY, int rY, vector<vector<int>> &v){
43         if(lY == rY){
44             if(lX == rX){
45                 seg[noX][noY] = v[rX][rY];
46             }else{
47                 seg[noX][noY] = f(seg[2*noX+1][noY],
48 seg[2*noX+2][noY]);
49             }
50         }else{
51             int m = (lY+rY)/2;
52
53             buildY(noX, lX, rX, 2*noY+1, lY, m, v);
54             buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);
55
56             seg[noX][noY] = f(seg[noX][2*noY+1], seg[noX][2*noY+2]);
57         }
58     }
59
60     void buildX(int noX, int lX, int rX, vector<
61 vector<int>> &v){
62         if(lX != rX){
63             int m = (lX+rX)/2;
64
65             buildX(2*noX+1, lX, m, v);

```

```

62         buildX(2*noX+2, m+1, rX, v);
63     }
64
65     buildY(noX, lX, rX, 0, 0, M - 1, v);
66 }
67
68 void updateY(int noX, int lX, int rX, int noY,
69 int lY, int rY, int y){
70     if(lY == rY){
71         if(lX == rX){
72             seg[noX][noY] = !seg[noX][noY];
73         }else{
74             seg[noX][noY] = seg[2*noX+1][noY] +
75 seg[2*noX+2][noY];
76         }
77     }else{
78         int m = (lY+rY)/2;
79
80         if(y <= m){
81             updateY(noX, lX, rX, 2*noY+1, lY, m, y);
82         }else if(m < y){
83             updateY(noX, lX, rX, 2*noY+2, m+1, rY, y);
84         }
85
86         seg[noX][noY] = seg[noX][2*noY+1] + seg[noX][2*noY+2];
87     }
88 }
89
90 void updateX(int noX, int lX, int rX, int x, int y){
91     int m = (lX+rX)/2;
92
93     if(lX != rX){
94         if(x <= m){
95             updateX(2*noX+1, lX, m, x, y);
96         }else if(m < x){
97             updateX(2*noX+2, m+1, rX, x, y);
98         }
99     }
100
101     updateY(noX, lX, rX, 0, 0, M - 1, y);
102 }
103
104 int queryY(int noX, int noY, int lY, int rY, int aY, int bY){
105     if(aY <= lY && rY <= bY) return seg[noX][noY];
106
107     int m = (lY+rY)/2;
108
109     if(bY <= m) return queryY(noX, 2*noY+1, lY, m, aY, bY);
110     if(m < aY) return queryY(noX, 2*noY+2, m+1, rY, aY, bY);
111
112     return f(queryY(noX, 2*noY+1, lY, m, aY, bY), queryY(noX, 2*noY+2, m+1, rY, aY, bY));
113 }
114
115 int queryX(int noX, int lX, int rX, int aX, int bX, int aY, int bY){
116     if(aX <= lX && rX <= bX) return queryY(noX, 0, 0, M - 1, aY, bY);
117
118     int m = (lX+rX)/2;
119
120     if(bX <= m) return queryX(2*noX+1, lX, m, aX, bX, aY, bY);
121     if(m < aX) return queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);

```

```

120         return f(queryX(2*noX+1, lX, m, aX, bX, aY,
121             bY), queryX(2*noX+2, m+1, rX, aX, bX, aY, bY));
122     }
123
124     void build(vector<vector<int>> &v) {
125         buildX(0, 0, N - 1, v);
126     }
127
128     int query(int aX, int aY, int bX, int bY) {
129         return queryX(0, 0, N - 1, aX, bX, aY, bY);
130     }
131
132     void update(int x, int y) {
133         updateX(0, 0, N - 1, x, y);
134     }
135 };

```

## 2.15 Lazy Addition To Segment

```

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 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 = -1; // change to -INF
   if there are negative numbers
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 };

```

## 2.16 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 = -1; // Change to -INF
    if there are negative numbers
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

```

```

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

```

## 2.17 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:
22 // O(log n) for both query and update
23
24 // How to use:
25 // MAX is the maximum index a node can assume
26 // Create a default null node
27 // Create a node to be the root of the segtree
28
29 // Segtree seg = Segtree(MAX);
30
31 const int MAX = 1e9+10;
32 const long long INF = 1e18+10;

```

```

27
28 typedef long long ftype;
29
30 struct Segtree {
31     vector<ftype> seg, d, e, lazy;
32     const ftype NEUTRAL = 0;
33     const ftype NEUTRAL_LAZY = -1; // change to -INF
34     // if the elements can be negative
35     int n;
36
37     Segtree(int n) {
38         this->n = n;
39         create();
40         create();
41     }
42
43     ftype apply_lazy(ftype a, ftype b, int len) {
44         if (b == NEUTRAL_LAZY) return a;
45         else return b * len; // change to a + b * len
46         // to add to an element instead of updating it
47     }
48
49     void propagate(int pos, int ini, int fim) {
50         if (seg[pos] == 0) return;
51
52         if (ini == fim) {
53             return;
54         }
55
56         int m = (ini + fim) >> 1;
57
58         if (e[pos] == 0) e[pos] = create();
59         if (d[pos] == 0) d[pos] = create();
60
61         lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[
62 pos], 1);
63         lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[
64 pos], 1);
65
66         seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[
67 pos], m - ini + 1);
68         seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[
69 pos], fim - m);
70
71         lazy[pos] = NEUTRAL_LAZY;
72     }
73
74     ftype f(ftype a, ftype b) {
75         return a + b;
76     }
77
78     ftype create() {
79         seg.push_back(0);
80         e.push_back(0);
81         d.push_back(0);
82         lazy.push_back(-1);
83         return seg.size() - 1;
84     }
85
86     ftype query(int pos, int ini, int fim, int p, int
87 q) {
88         propagate(pos, ini, fim);
89         if (q < ini || p > fim) return NEUTRAL;
90         if (pos == 0) return 0;
91         if (p <= ini && fim <= q) return seg[pos];
92         int m = (ini + fim) >> 1;
93         return f(query(e[pos], ini, m, p, q), query(d
94 [pos], m + 1, fim, p, q));
95     }
96
97     void update(int pos, int ini, int fim, int p, int
98 q, int val) {
99         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             ;
98             seg[pos] = apply_lazy(seg[pos], val, fim
99             - ini + 1);
100         }
101         return;
102
103         int m = (ini + fim) >> 1;
104
105         if (e[pos] == 0) e[pos] = create();
106         update(e[pos], ini, m, p, q, val);
107
108         if (d[pos] == 0) d[pos] = create();
109         update(d[pos], m + 1, fim, p, q, val);
110
111         seg[pos] = f(seg[e[pos]], seg[d[pos]]);
112     }
113
114     ftype query(int p, int q) {
115         return query(1, 1, n, p, q);
116     }
117
118     void update(int p, int q, int val) {
119         update(1, 1, n, p, q, val);
120     }

```

## 2.18 Persistent

```

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

```

```

32
33 Segtree(int n) {
34     this->n = n;
35 }
36
37 ftype f(ftype a, ftype b) {
38     return a + b;
39 }
40
41 ftype create() {
42     seg.push_back(0);
43     e.push_back(0);
44     d.push_back(0);
45     return seg.size() - 1;
46 }
47
48 ftype query(int pos, int ini, int fim, int p, int
49 q) {
50     if (q < ini || p > fim) return NEUTRAL;
51     if (pos == 0) return 0;
52     if (p <= ini && fim <= q) return seg[pos];
53     int m = (ini + fim) >> 1;
54     return f(query(e[pos], ini, m, p, q), query(d
55 [pos], m + 1, fim, p, q));
56 }
57
58 int update(int pos, int ini, int fim, int id, int
59 val) {
60     int novo = create();
61
62     seg[novo] = seg[pos];
63     e[novo] = e[pos];
64     d[novo] = d[pos];
65
66     if (ini == fim) {
67         seg[novo] = val;
68         return novo;
69     }
70
71     int m = (ini + fim) >> 1;
72
73     if (id <= m) e[novo] = update(e[novo], ini, m
74 , id, val);
75     else d[novo] = update(d[novo], m + 1, fim, id
76 , val);
77
78     seg[novo] = f(seg[e[novo]], seg[d[novo]]);
79
80     return novo;
81 }
82
83 ftype query(int pos, int p, int q) {
84     return query(pos, 1, n, p, q);
85 }
86
87 int update(int pos, int id, int val) {
88     return update(pos, 1, n, id, val);
89 }
90
91 };

```

## 3 Strings

### 3.1 Lcs

```

1 // Description:
2 // Finds the longest common subsequence between two
3 // string
4
5 // Problem:
6 // https://codeforces.com/gym/103134/problem/B
7
8 // Complexity:

```



```

8 // 0(mn) where m and n are the length of the strings
9
10 string lcsAlgo(string s1, string s2, int m, int n) {
11     int LCS_table[m + 1][n + 1];
12
13     for (int i = 0; i <= m; i++) {
14         for (int j = 0; j <= n; j++) {
15             if (i == 0 || j == 0)
16                 LCS_table[i][j] = 0;
17             else if (s1[i - 1] == s2[j - 1])
18                 LCS_table[i][j] = LCS_table[i - 1][j - 1] +
19 1;
20             else
21                 LCS_table[i][j] = max(LCS_table[i - 1][j],
22 LCS_table[i][j - 1]);
23         }
24     }
25
26     int index = LCS_table[m][n];
27     char lcsAlgo[index + 1];
28     lcsAlgo[index] = '\0';
29
30     int i = m, j = n;
31     while (i > 0 && j > 0) {
32         if (s1[i - 1] == s2[j - 1]) {
33             lcsAlgo[index - 1] = s1[i - 1];
34             i--;
35             j--;
36             index--;
37         }
38         else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
39             i--;
40         else
41             j--;
42     }
43
44     return lcsAlgo;
45 }

```

### 3.2 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 }

```

### 3.3 Hash2

```

1 // Hashed String {{{
2 class HashedString {
3     static const int M = (1LL << 61) - 1;
4     static const int B;
5     static vector<int> pow;
6
7     int N;
8     vector<int> p_hash;
9
10     __int128 mul(int a, int b) { return (__int128)a * b; }
11     int mod_mul(int a, int b) { return mul(a, b) % M; }

```

```

12 public:
13     explicit HashedString(string const& s) {
14         while (size(pow) < size(s) + 1) pow.push_back(
15             mod_mul(pow.back(), B));
16
17         p_hash.resize(size(s) + 1);
18         p_hash[0] = 0;
19         for (int i = 0; i < size(s); i++)
20             p_hash[i + 1] = (mul(p_hash[i], B) + s[i]) % M;
21     }
22
23     int get_hash(int l, int r) {
24         int raw_val = p_hash[r + 1] - mod_mul(p_hash[l],
25 pow[r - l + 1]);
26         return (raw_val + M) % M;
27     }
28
29     int prefix(int len) { return get_hash(0, len - 1); }
30     int suffix(int len) { return get_hash(N - len, N - 1); }
31 }
32
33 int whole() { return get_hash(0, N - 1); }
34 int substr(int l, int len) {
35     int r = l + len - 1;
36     r = min(r, N - 1);
37     return get_hash(l, r);
38 }
39
40 vector<int> HashedString::pow{1};
41 mt19937 rng((uint32_t)chrono::steady_clock::now().
42 time_since_epoch().count());
43 const int HashedString::B = uniform_int_distribution<
44 int>(0, M - 1)(rng);
45 //}}}

```

### 3.4 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 }

```

### 3.5 Generate All Permutations

```

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

```

### 3.6 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 }

```

### 3.7 Hash

```

1 // Description:
2 // Turns a string into a integer.
3 // If the hash is different then the strings are
4 // different.
5 // If the hash is the same the strings may be
6 // different.
7
8 // Problem:
9 // https://codeforces.com/gym/104518/problem/I
10
11 // Complexity:
12 // O(n) to calculate the hash
13 // O(1) to query
14
15 // Notes:
16 // Primes 1000000007, 1000041323, 100663319,
17 // 201326611, 1000015553, 1000028537
18
19 struct Hash {

```

```

17     const ll P = 31;
18     int n; string s;
19     vector<ll> h, hi, p;
20     Hash() {}
21     Hash(string s): s(s), n(s.size()), h(n), hi(n), p
22     (n) {
23         for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1)
24         % MOD;
25         for (int i=0;i<n;i++)
26             h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
27         for (int i=n-1;i>=0;i--)
28             hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
29             % MOD;
30     }
31     int query(int l, int r) {
32         ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]:0)%MOD :
33         0));
34         return hash < 0 ? hash + MOD : hash;
35     }
36
37     int query_inv(int l, int r) {
38         ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
39         +1]:0)%MOD : 0));
40         return hash < 0 ? hash + MOD : hash;
41     }
42 }

```

### 3.8 Generate All Sequences Length K

```

1 // gera todas as ípossveis @sequencias usando as letras
2 // em set (de comprimento n) e que tenham tamanho k
3 // sequence = ""
4 // generate_sequences(char set[], string
5 // sequence, int n, int k) {
6 //     if (k == 0){
7 //         return { sequence };
8 //     }
9 //     vector<string> ans;
10 //     for (int i = 0; i < n; i++) {
11 //         auto aux = generate_sequences(set, sequence +
12 //         set[i], n, k - 1);
13 //         ans.insert(ans.end(), aux.begin(), aux.end())
14 //     };
15 //     // for (auto e : aux) ans.push_back(e);
16 // }
17
18 // return ans;
19 }

```

### 3.9 Suffix Array

```

1 // Description:
2 // Suffix array is an array with the indexes of the
3 // starting letter of every
4 // suffix in an array sorted in lexicographical order
5 // .
6 // Problem:
7 // https://codeforces.com/edu/course/2/lesson/2/1/
8 // practice/contest/269100/problem/A
9
10 // Complexity:
11 // O(n log n) with radix sort
12 // O(n log ~ 2 n) with regular sort
13
14 // Notes:
15 // Relevant Problems
16 // Substring search: Queries to know whether a given
17 // substring is present in a string
18 // Binary search for the first suffix that is greater
19 // or equal
20 // O(log n |p|) where |p| is the total size of the
21 // substrings queried

```

```

17 //
18 // Substring size: Queries to know how many times a
    given substring appears in a string
19 // Binary search both for first and last that is
    greater or equal
20 //
21 // Number of different substrings:
22 // A given suffix gives sz new substrings being sz
    the size of the suffix
23 // We can subtract the lcp (longest common prefix) to
    remove substrings
24 // that were already counted.
25 //
26 // Longest common substring between two strings:
27 // We can calculate the suffix array and lcp array of
    the two strings
28 // concatenated with a character greater than $ and
    smaller than A (like '&')
29 // The answer will be the lcp between two consecutive
    suffixes that belong to different strings
30 // (index at suffix array <= size of the first array)
31
32 void radix_sort(vector<pair<pair<int, int>, int>>& a)
    {
33     int n = a.size();
34     vector<pair<pair<int, int>, int>> ans(n);
35
36     vector<int> count(n);
37
38     for (int i = 0; i < n; i++) {
39         count[a[i].first.second]++;
40     }
41
42     vector<int> p(n);
43
44     p[0] = 0;
45     for (int i = 1; i < n; i++) {
46         p[i] = p[i - 1] + count[i - 1];
47     }
48
49     for (int i = 0; i < n; i++) {
50         ans[p[a[i].first.second]++] = a[i];
51     }
52
53     a = ans;
54
55     count.assign(n, 0);
56
57     for (int i = 0; i < n; i++) {
58         count[a[i].first.first]++;
59     }
60
61     p.assign(n, 0);
62
63     p[0] = 0;
64     for (int i = 1; i < n; i++) {
65         p[i] = p[i - 1] + count[i - 1];
66     }
67
68     for (int i = 0; i < n; i++) {
69         ans[p[a[i].first.first]++] = a[i];
70     }
71
72     a = ans;
73 }
74
75 vector<int> p, c;
76
77 vector<int> suffix_array(string s) {
78     int n = s.size();
79     vector<pair<char, int>> a(n);
80     p.assign(n, 0);
81     c.assign(n, 0);
82
83     for (int i = 0; i < n; i++) {
84         a[i] = mp(s[i], i);
85     }
86
87     sort(a.begin(), a.end());
88
89     for (int i = 0; i < n; i++) {
90         p[i] = a[i].second;
91     }
92
93     c[p[0]] = 0;
94     for (int i = 1; i < n; i++) {
95         if (a[i].first == a[i - 1].first) c[p[i]] = c[p[i]
            - 1];
96         else c[p[i]] = c[p[i - 1]] + 1;
97     }
98
99     int k = 0;
100     while ((1 << k) < n) {
101         vector<pair<pair<int, int>, int>> a(n);
102         for (int i = 0; i < n; i++) {
103             a[i] = mp(mp(c[i], c[(i + (1 << k)) % n]), i);
104         }
105
106         radix_sort(a);
107
108         for (int i = 0; i < n; i++) {
109             p[i] = a[i].second;
110         }
111
112         c[p[0]] = 0;
113         for (int i = 1; i < n; i++) {
114             if (a[i].first == a[i - 1].first) c[p[i]] = c[p[i]
                - 1];
115             else c[p[i]] = c[p[i - 1]] + 1;
116         }
117
118         k++;
119     }
120
121     /* for (int i = 0; i < n; i++) {
122         for (int j = p[i]; j < n; j++) {
123             cout << s[j];
124         }
125         cout << '\n';
126     } */
127
128     return p;
129 }
130
131 // the first suffix will always be $ the (n - 1)th
    character in the string
132 vector<int> lcp_array(string s) {
133     int n = s.size();
134     vector<int> ans(n);
135     // minimum lcp
136     int k = 0;
137     for (int i = 0; i < n - 1; i++) {
138         // indice in the suffix array p of suffix
            starting in i
139         int pi = c[i];
140         // start index of the previous suffix in suffix
            array
141         int j = p[pi - 1];
142         while (s[i + k] == s[j + k]) k++;
143         ans[pi] = k;
144         k = max(k - 1, 0);
145     }
146
147     return ans;
148 }

```

## 4 Algorithms

### 4.1 Lcs

```
1 // Longest Common Subsequence
2 //
3 // Computa a LCS entre dois arrays usando
4 // o algoritmo de Hirschberg para recuperar
5 //
6 // O(n*m), O(n+m) de memoria
7
8 int lcs_s[MAX], lcs_t[MAX];
9 int dp[2][MAX];
10
11 // dp[0][j] = max lcs(s[li...ri], t[lj, lj+j])
12 void dp_top(int li, int ri, int lj, int rj) {
13     memset(dp[0], 0, (rj-lj+1)*sizeof(dp[0][0]));
14     for (int i = li; i <= ri; i++) {
15         for (int j = rj; j >= lj; j--)
16             dp[0][j - lj] = max(dp[0][j - lj],
17 (lcs_s[i] == lcs_t[j]) + (j > lj ? dp[0][j-1 -
lj] : 0));
18         for (int j = lj+1; j <= rj; j++)
19             dp[0][j - lj] = max(dp[0][j - lj], dp[0][j-1 -
lj]);
20     }
21 }
22
23 // dp[1][j] = max lcs(s[li...ri], t[lj+j, rj])
24 void dp_bottom(int li, int ri, int lj, int rj) {
25     memset(dp[1], 0, (rj-lj+1)*sizeof(dp[1][0]));
26     for (int i = ri; i >= li; i--) {
27         for (int j = lj; j <= rj; j++)
28             dp[1][j - lj] = max(dp[1][j - lj],
29 (lcs_s[i] == lcs_t[j]) + (j < rj ? dp[1][j+1 -
lj] : 0));
30         for (int j = rj-1; j >= lj; j--)
31             dp[1][j - lj] = max(dp[1][j - lj], dp[1][j+1 -
lj]);
32     }
33 }
34
35 void solve(vector<int>& ans, int li, int ri, int lj,
int rj) {
36     if (li == ri){
37         for (int j = lj; j <= rj; j++)
38             if (lcs_s[li] == lcs_t[j]){
39                 ans.push_back(lcs_t[j]);
40                 break;
41             }
42         return;
43     }
44     if (lj == rj){
45         for (int i = li; i <= ri; i++){
46             if (lcs_s[i] == lcs_t[lj]){
47                 ans.push_back(lcs_s[i]);
48                 break;
49             }
50         }
51         return;
52     }
53     int mi = (li+ri)/2;
54     dp_top(li, mi, lj, rj), dp_bottom(mi+1, ri, lj, rj)
55     ;
56     int j_ = 0, mx = -1;
57
58     for (int j = lj-1; j <= rj; j++) {
59         int val = 0;
60         if (j >= lj) val += dp[0][j - lj];
61         if (j < rj) val += dp[1][j+1 - lj];
62     }
```

```
63         if (val >= mx) mx = val, j_ = j;
64     }
65     if (mx == -1) return;
66     solve(ans, li, mi, lj, j_), solve(ans, mi+1, ri, j_
+1, rj);
67 }
68
69 vector<int> lcs(const vector<int>& s, const vector<
int>& t) {
70     for (int i = 0; i < s.size(); i++) lcs_s[i] = s[i];
71     for (int i = 0; i < t.size(); i++) lcs_t[i] = t[i];
72     vector<int> ans;
73     solve(ans, 0, s.size()-1, 0, t.size()-1);
74     return ans;
75 }
```

### 4.2 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     }
```

```

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 }

```

### 4.3 Lis

```

1 // Returns the size of the sequence
2 int lis(vector<int> const& a) {
3     int n = a.size();
4     vector<int> d(n, 1);
5     for (int i = 0; i < n; i++) {
6         for (int j = 0; j < i; j++) {
7             if (a[j] < a[i])
8                 d[i] = max(d[i], d[j] + 1);
9         }
10    }
11
12    int ans = d[0];
13    for (int i = 1; i < n; i++) {
14        ans = max(ans, d[i]);
15    }
16    return ans;
17 }
18
19 // Returns the sequence
20 template<typename T> vector<T> lis(vector<T>& v) {
21     int n = v.size(), m = -1;
22     vector<T> d(n+1, INF);
23     vector<int> l(n);
24     d[0] = -INF;
25
26     for (int i = 0; i < n; i++) {
27         // Para non-decreasing use upper_bound()
28         int t = lower_bound(d.begin(), d.end(), v[i]) - d
29             .begin();
30         d[t] = v[i], l[i] = t, m = max(m, t);
31     }
32
33     int p = n;
34     vector<T> ret;
35     while (p--) if (l[p] == m) {
36         ret.push_back(v[p]);
37         m--;
38     }
39     reverse(ret.begin(), ret.end());
40
41     return ret;
42 }

```

### 4.4 Binary Search First True

```

1 int first_true(int lo, int hi, function<bool(int)> f)
2 {
3     hi++;
4     while (lo < hi) {
5         int mid = lo + (hi - lo) / 2;
6         if (f(mid)) {
7             hi = mid;
8         } else {
9             lo = mid + 1;
10        }
11    }
12    return lo;
13 }

```

### 4.5 Subsets

```

1 void subsets(vector<int>& nums){
2     int n = nums.size();
3     int powSize = 1 << n;
4
5     for(int counter = 0; counter < powSize; counter++){
6         for(int j = 0; j < n; j++){
7             if((counter & (1LL << j)) != 0) {
8                 cout << nums[j] << ' ';
9             }
10        }
11        cout << '\n';
12    }
13 }

```

### 4.6 Ternary Search

```

1 double ternary_search(double l, double r) {
2     double eps = 1e-9; //set the error
3     limit here
4     while (r - l > eps) {
5         double m1 = l + (r - l) / 3;
6         double m2 = r - (r - l) / 3;
7         double f1 = f(m1); //evaluates the
8         function at m1
9         double f2 = f(m2); //evaluates the
10        function at m2
11        if (f1 < f2)
12            l = m1;
13        else
14            r = m2;
15    }
16    return f(l); //return the
17    maximum of f(x) in [l, r]
18 }

```

### 4.7 Binary Search Last True

```

1 int last_true(int lo, int hi, function<bool(int)> f)
2 {
3     lo--;
4     while (lo < hi) {
5         int mid = lo + (hi - lo + 1) / 2;
6         if (f(mid)) {
7             lo = mid;
8         } else {
9             hi = mid - 1;
10        }
11    }
12    return lo;
13 }

```

### 4.8 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 }

```

## 4.9 Binary Search Real

```

1  int cnt = 100;
2  ld l = 1e-7, r = 1e6+10;
3  while (cnt--) {
4      ld mid = (l + r) / 2;
5
6      if (f(mid)) {
7          l = mid;
8      } else {
9          r = mid;
10     }
11 }

```

## 5 Template

### 5.1 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);
6      cin.tie(NULL);
7  #define vi vector<int>
8  #define ll long long
9  #define pb push_back
10 #define mp make_pair
11 #define ff first
12 #define ss second
13 #define pii pair<int, int>
14 #define MOD 1000000007
15 #define sqr(x) ((x) * (x))
16 #define all(x) (x).begin(), (x).end()
17 #define FOR(i, j, n) for (int i = j; i < n; i++)
18 #define qle(i, n) (i == n ? "\n" : " ")
19 #define endl "\n"
20 const int oo = 1e9;
21 const int MAX = 1e6;
22
23 int32_t main(){ optimize;
24
25     return 0;
26 }

```

## 5.2 Template Clean

```

1  // Notes:
2  // Compile and execute
3  // g++ teste.cpp -o teste -std=c++17
4  // ./teste < teste.txt
5
6  // Print with precision
7  // cout << fixed << setprecision(12) << value << endl
8      ;
9
10 // File as input and output
11 // freopen("input.txt", "r", stdin);
12 // freopen("output.txt", "w", stdout);
13
14 #include <bits/stdc++.h>
15 using namespace std;
16
17 #define pb push_back
18 #define mp make_pair
19 #define mt make_tuple
20 #define ff first
21 #define ss second
22 #define ld long double
23 #define ll long long
24 #define int long long
25 #define pii pair<int, int>
26 #define tii tuple<int, int, int>
27
28 int main() {
29     ios::sync_with_stdio(false);
30     cin.tie(NULL);
31
32
33     return 0;
34 }

```

## 6 Math

### 6.1 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 }

```

### 6.2 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  #define popcount(x) __builtin_popcountll((unsigned ll
6      )x)
7  // turn bit off
8
9  bool bit_on(int n, int bit) {
10     if(1 & (n >> bit)) return true;
11     else return false;
12 }

```

## 6.3 To Decimal

```
1 const string digits { "0123456789
   ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
2
3 long long to_decimal(const string& rep, long long
   base) {
4     long long n = 0;
5
6     for (auto c : rep) {
7         // if the number can't be represented in this
       base
8         if (c > digits[base - 1]) return -1;
9         n *= base;
10        n += digits.find(c);
11    }
12
13    return n;
14 }
```

## 6.4 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 }
```

## 6.5 Fft

```
1 / FFT {{{
2 using cd = complex<double>;
3 const double PI = acos(-1);
4
5 void fft(vector<cd> &A, bool invert) {
6     int N = size(A);
7
8     for (int i = 1, j = 0; i < N; i++) {
9         int bit = N >> 1;
10        for (; j & bit; bit >>= 1)
11            j ^= bit;
12        j ^= bit;
13
14        if (i < j)
15            swap(A[i], A[j]);
16    }
17
18    for (int len = 2; len <= N; len <= 1) {
19        double ang = 2 * PI / len * (invert ? -1 : 1);
20        cd wlen(cos(ang), sin(ang));
21        for (int i = 0; i < N; i += len) {
```

```
22            cd w(1);
23            for (int j = 0; j < len/2; j++) {
24                cd u = A[i+j], v = A[i+j+len/2] * w;
25                A[i+j] = u + v;
26                A[i+j+len/2] = u - v;
27                w *= wlen;
28            }
29        }
30    }
31
32    if (invert) {
33        for (auto &x : A)
34            x /= N;
35    }
36 }
37
38 vector<int> multiply(vector<int> const& A, vector<int>
   > const& B) {
39     vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B)
   ));
40     int N = 1;
41     while (N < size(A) + size(B))
42         N <<= 1;
43     fa.resize(N);
44     fb.resize(N);
45
46     fft(fa, false);
47     fft(fb, false);
48     for (int i = 0; i < N; i++)
49         fa[i] *= fb[i];
50     fft(fa, true);
51
52     vector<int> result(N);
53     for (int i = 0; i < N; i++)
54         result[i] = round(fa[i].real());
55     return result;
56 }
57 // }}}}
```

## 6.6 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 *
       tan(x) + t * x * x + u;
7 }
8
9 ld root(ld a, ld b) {
10     while (b - a >= EPS1) {
11         ld c = (a + b) / 2.0;
12         ld y = f(c);
13
14         if (y < 0) b = c;
15         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 <<
       setprecision(4) << ans << '\n';
24     else cout << "No solution\n";
25
26     return 0;
27 }
```

## 6.7 Set Operations

```

1 // Complexity;
2 // O(n * m) being n and m the sizes of the two sets
3 // 2*(count1+count2)-1 (where countX is the distance
   between firstX and lastX):
4
5 vector<int> res;
6 set_union(s1.begin(), s1.end(), s2.begin(), s2.end(),
   inserter(res, res.begin()));
7 set_intersection(s1.begin(), s1.end(), s2.begin(), s2
   .end(), inserter(res, res.begin()));
8 // present in the first set, but not in the second
9 set_difference(s1.begin(), s1.end(), s2.begin(), s2.
   end(), inserter(res, res.begin()));
10 // present in one of the sets, but not in the other
11 set_symmetric_difference(s1.begin(), s1.end(), s2.
   begin(), s2.end(), inserter(res, res.begin()));

```

## 6.8 Subsets

```

1 void subsets(vector<int>& nums){
2     int n = nums.size();
3     int powSize = 1 << n;
4
5     for(int counter = 0; counter < powSize; counter++){
6         for(int j = 0; j < n; j++) {
7             if((counter & (1LL << j)) != 0) {
8                 cout << nums[j] << ' ';
9             }
10            cout << '\n';
11        }
12    }
13 }

```

## 6.9 Pascalsrule Stifel

```

1 // Description:
2 // Calculates a binomial n chooses k based on the
   value of a previous binomial.
3
4 // Complexity:
5 // O(n * k)
6
7 vector<vector<int>>> comb(MAX + 1, vector<int>(MAX +
   1, 0));
8
9 for (int n = 0; n <= MAX; n++) {
10     comb[n][0] = 1;
11 }
12
13 for (int n = 1; n <= MAX; n++) {
14     for (int k = 1; k <= n; k++) {
15         comb[n][k] = comb[n - 1][k - 1] + comb[n - 1][k];
16     }
17 }

```

## 6.10 Sieve Of Eratosthenes

```

1 vector<bool> is_prime(MAX, true);
2 vector<int> primes;
3
4 void sieve() {
5     is_prime[0] = is_prime[1] = false;
6     for (int i = 2; i < MAX; i++) {
7         if (is_prime[i]) {
8             primes.push_back(i);
9
10            for (int j = i + i; j < MAX; j += i)
11                is_prime[j] = false;
12        }
13    }
14 }

```

## 6.11 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,
   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
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 }

```

## 6.12 Mobius

```

1 vector<int> m(MAXN, 0), lp(MAXN, 0);
2 m[1] = 1;
3 for (int i = 2; i < MAXN; ++i) {
4     if (!lp[i]) for (int j = i; j < MAXN; j += i)
5         if (!lp[j]) lp[j] = i;
6     m[i] = [&](int x) {
7         int cnt = 0;
8         while (x > 1) {
9             int k = 0, d = lp[x];
10            while (x % d == 0) {
11                x /= d;
12                ++k;
13                if (k > 1) return 0;
14            }
15            ++cnt;
16        }
17        if (cnt & 1) return -1;
18        return 1;
19    }(i);
20 }

```

## 6.13 Representation Arbitrary Base

```

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

```

```

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 }

```

## 6.14 Horner Algorithm

```

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

```

## 6.15 Prime Factors

```

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

```

## 6.16 Binary To Decimal

```

1 int binary_to_decimal(long long n) {
2     int dec = 0, i = 0, rem;
3
4     while (n != 0) {
5         rem = n % 10;
6         n /= 10;
7         dec += rem * pow(2, i);
8         ++i;
9     }
10

```

```

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

```

## 6.17 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
24 // ans 6 ways using 2 equal colors and 1 distinct one
25
26 // Adding another column, there are:
27 // 3 ways to go from 2 equal to 2 equal
28 // 2 ways to go from 2 equal to 3 distinct
29 // 2 ways to go from 3 distinct to 2 equal
30 // 2 ways to go from 3 distinct to 3 distinct
31
32 // So we start with matrix 6 6 and multiply it by the
33 // transition 3 2 and get 18 12
34
35 //
36 //      6 6
37 //      2 2      12 12
38 // the we can exponentiate this matrix to find the
39 // nth column
40
41 // Problem:
42 // https://cses.fi/problemset/task/1722/
43
44 // Complexity:
45 // O(log n)
46
47 // How to use:
48 // vector<vector<ll>> v = {{1, 1}, {1, 0}};
49 // Matriz transition = Matriz(v);
50 // cout << fexp(transition, n)[0][1] << '\n';
51
52 using ll = long long;
53
54 const int MOD = 1e9+7;
55
56 struct Matriz{

```

```

48     vector<vector<ll>> mat;
49     int rows, columns;
50
51     vector<ll> operator[](int i){
52         return mat[i];
53     }
54
55     Matriz(vector<vector<ll>>& matriz){
56         mat = matriz;
57         rows = mat.size();
58         columns = mat[0].size();
59     }
60
61     Matriz(int row, int column, bool identity=false){
62         rows = row; columns = column;
63         mat.assign(rows, vector<ll>(columns, 0));
64         if(identity) {
65             for(int i = 0; i < min(rows, columns); i
66             ++){
67                 mat[i][i] = 1;
68             }
69         }
70
71     Matriz operator * (Matriz a) {
72         assert(columns == a.rows);
73         vector<vector<ll>> resp(rows, vector<ll>(a.
74         columns, 0));
75
76         for(int i = 0; i < rows; i++){
77             for(int j = 0; j < a.columns; j++){
78                 for(int k = 0; k < a.rows; k++){
79                     resp[i][j] = (resp[i][j] + (mat[i
80                     ][k] * 1LL * a[k][j]) % MOD) % MOD;
81                 }
82             }
83         }
84         return Matriz(resp);
85     }
86
87     Matriz operator + (Matriz a) {
88         assert(rows == a.rows && columns == a.columns
89         );
90         vector<vector<ll>> resp(rows, vector<ll>(
91         columns, 0));
92         for(int i = 0; i < rows; i++){
93             for(int j = 0; j < columns; j++){
94                 resp[i][j] = (resp[i][j] + mat[i][j]
95                 + a[i][j]) % MOD;
96             }
97         }
98         return Matriz(resp);
99     }
100 }
101
102 Matriz fexp(Matriz base, ll exponent){
103     Matriz result = Matriz(base.rows, base.columns, 1);
104     while(exponent > 0){
105         if(exponent & 1LL) result = result * base;
106         base = base * base;
107         exponent = exponent >> 1;
108     }
109     return result;
110 }

```

## 6.18 Fast Exponentiation

```

1 ll fexp(ll b, ll e, ll mod) {
2     ll res = 1;
3     b %= mod;
4     while(e){
5         if(e & 1LL)
6             res = (res * b) % mod;

```

```

7         e = e >> 1LL;
8         b = (b * b) % mod;
9     }
10    return res;
11 }

```

## 6.19 Divisors

```

1 vector<long long> all_divisors(long long n) {
2     vector<long long> ans;
3     for(long long a = 1; a*a <= n; a++){
4         if(n % a == 0) {
5             long long b = n / a;
6             ans.push_back(a);
7             if(a != b) ans.push_back(b);
8         }
9     }
10    sort(ans.begin(), ans.end());
11    return ans;
12 }

```

## 6.20 Ntt

```

1 // Aritmetica Modular
2 //
3 // 0 mod tem q ser primo
4
5 template<int p> struct mod_int {
6     ll expo(ll b, ll e) {
7         ll ret = 1;
8         while (e) {
9             if (e % 2) ret = ret * b % p;
10            e /= 2, b = b * b % p;
11        }
12        return ret;
13    }
14    ll inv(ll b) { return expo(b, p-2); }
15
16    using m = mod_int;
17    int v;
18    mod_int() : v(0) {}
19    mod_int(ll v_) {
20        if (v_ >= p or v_ <= -p) v_ %= p;
21        if (v_ < 0) v_ += p;
22        v = v_;
23    }
24    m& operator +=(const m& a) {
25        v += a.v;
26        if (v >= p) v -= p;
27        return *this;
28    }
29    m& operator -=(const m& a) {
30        v -= a.v;
31        if (v < 0) v += p;
32        return *this;
33    }
34    m& operator *=(const m& a) {
35        v = v * ll(a.v) % p;
36        return *this;
37    }
38    m& operator /=(const m& a) {
39        v = v * inv(a.v) % p;
40        return *this;
41    }
42    m operator -(){ return m(-v); }
43    m& operator ^=(ll e) {
44        if (e < 0) {
45            v = inv(v);
46            e = -e;
47        }
48        v = expo(v, e);
49        // possivel otimizacao:

```

```

50        // cuidado com 0^0
51        // v = expo(v, e%(p-1));
52        return *this;
53    }
54    bool operator ==(const m& a) { return v == a.v; }
55    bool operator !=(const m& a) { return v != a.v; }
56
57    friend istream& operator >>(istream& in, m& a) {
58        ll val; in >> val;
59        a = m(val);
60        return in;
61    }
62    friend ostream& operator <<(ostream& out, m a) {
63        return out << a.v;
64    }
65    friend m operator +(m a, m b) { return a += b; }
66    friend m operator -(m a, m b) { return a -= b; }
67    friend m operator *(m a, m b) { return a *= b; }
68    friend m operator /(m a, m b) { return a /= b; }
69    friend m operator ^(m a, ll e) { return a ^= e; }
70 };
71
72 typedef mod_int<(int)1e9+7> mint;
73
74 // NTT
75 //
76 // Precisa do mint (primitivas de aritmetica modular)
77 //
78 // O(n log (n))
79
80 const int MOD = 998244353;
81 typedef mod_int<MOD> mint;
82
83 void ntt(vector<mint>& a, bool rev) {
84     int n = a.size(); auto b = a;
85     assert(!(n&(n-1)));
86     mint g = 1;
87     while ((g^(MOD / 2)) == 1) g += 1;
88     if (rev) g = 1 / g;
89
90     for (int step = n / 2; step; step /= 2) {
91         mint w = g^(MOD / (n / step)), wn = 1;
92         for (int i = 0; i < n/2; i += step) {
93             for (int j = 0; j < step; j++) {
94                 auto u = a[2 * i + j], v = wn * a[2 * i + j +
95                     step];
96                 b[i+j] = u + v; b[i + n/2 + j] = u - v;
97             }
98             wn = wn * w;
99         }
100        swap(a, b);
101    }
102    if (rev) {
103        auto n1 = mint(1) / n;
104        for (auto& x : a) x *= n1;
105    }
106 }
107
108 vector<mint> convolution(const vector<mint>& a, const
109     vector<mint>& b) {
110     vector<mint> l(a.begin(), a.end()), r(b.begin(), b.
111         end());
112     int N = l.size()+r.size()-1, n = 1;
113     while (n <= N) n *= 2;
114     l.resize(n);
115     r.resize(n);
116     ntt(l, false);
117     ntt(r, false);
118     for (int i = 0; i < n; i++) l[i] *= r[i];
119     ntt(l, true);
120     l.resize(N);
121     return l;
122 }

```

## 6.21 Phi

```

1 // Description:
2 // Euler's totient function.
3 // phi(n) is the amount of numbers in the range (1, n
   ) that are coprime with n
4
5 // Complexity:
6 // phi(n) - sqrt(n)
7 // phi of all numbers from 1 to n - O (n log log n)
8
9 // Properties:
10 // phi(p ^ k) = p ^ k - p ^ (k - 1)
11 // phi(p) = p - 1
12 // phi(ab) = phi(a) * phi(b) * d / phi(d) being d =
   gcd(a, b)
13
14 int phi(int n) {
15     int result = n;
16     for (int i = 2; i * i <= n; i++) {
17         if (n % i == 0) {
18             while (n % i == 0)
19                 n /= i;
20             result -= result / i;
21         }
22     }
23     if (n > 1)
24         result -= result / n;
25     return result;
26 }
27
28 void phi_1_to_n(int n) {
29     vector<int> phi(n + 1);
30     for (int i = 0; i <= n; i++)
31         phi[i] = i;
32
33     for (int i = 2; i <= n; i++) {
34         if (phi[i] == i) {
35             for (int j = i; j <= n; j += i)
36                 phi[j] -= phi[j] / i;
37         }
38     }
39 }

```

```

26 100/20 = 5
27 100/21 = 4
28 100/22 = 4
29 100/23 = 4
30 100/24 = 4
31 100/25 = 4
32 100/26 = 3
33 100/27 = 3
34 100/28 = 3
35 100/29 = 3
36 100/30 = 3
37 100/31 = 3
38 100/32 = 3
39 100/33 = 3
40 100/34 = 2
41 100/35 = 2
42 100/36 = 2
43 100/37 = 2
44 100/38 = 2
45 100/39 = 2
46 100/40 = 2
47 100/41 = 2
48 100/42 = 2
49 100/43 = 2
50 100/44 = 2
51 100/45 = 2
52 100/46 = 2
53 100/47 = 2
54 100/48 = 2
55 100/49 = 2
56 100/50 = 2
57 100/51 = 1
58 100/52 = 1
59 100/53 = 1
60 100/54 = 1
61 100/55 = 1
62 100/56 = 1
63 100/57 = 1
64 100/58 = 1
65 100/59 = 1
66 100/60 = 1
67 100/61 = 1
68 100/62 = 1
69 100/63 = 1
70 100/64 = 1
71 100/65 = 1
72 100/66 = 1
73 100/67 = 1
74 100/68 = 1
75 100/69 = 1
76 100/70 = 1
77 100/71 = 1
78 100/72 = 1
79 100/73 = 1
80 100/74 = 1
81 100/75 = 1
82 100/76 = 1
83 100/77 = 1
84 100/78 = 1
85 100/79 = 1
86 100/80 = 1
87 100/81 = 1
88 100/82 = 1
89 100/83 = 1
90 100/84 = 1
91 100/85 = 1
92 100/86 = 1
93 100/87 = 1
94 100/88 = 1
95 100/89 = 1
96 100/90 = 1
97 100/91 = 1
98 100/92 = 1

```

## 6.22 Division Trick

```

1 for(int l = 1, r; l <= n; l = r + 1) {
2     r = n / (n / l);
3     // n / i has the same value for l <= i <= r
4     // from l to r, n / i has the same value which is
   n / l
5 }
6
7 /* 100/1 = 100
8 100/2 = 50
9 100/3 = 33
10 100/4 = 25
11 100/5 = 20
12 100/6 = 16
13 100/7 = 14
14 100/8 = 12
15 100/9 = 11
16 100/10 = 10
17 100/11 = 9
18 100/12 = 8
19 100/13 = 7
20 100/14 = 7
21 100/15 = 6
22 100/16 = 6
23 100/17 = 5
24 100/18 = 5
25 100/19 = 5

```

```

99 100/93 = 1
100 100/94 = 1
101 100/95 = 1
102 100/96 = 1
103 100/97 = 1
104 100/98 = 1
105 100/99 = 1 */

```

## 6.23 Ceil

```

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

```

## 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 || j==0)
7                 dp[i][j] = 0;
8             else if(peso[i-1]<=j)
9                 dp[i][j] = max(val[i-1]+dp[i-1][j-
10                     peso[i-1]], dp[i-1][j]);
11             else
12                 dp[i][j] = dp[i-1][j];
13         }
14     }
15     return dp[n][m];
16 }

```

### 7.2 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 }

```

### 7.3 Substr Palindrome

```

1 // êvoc deve informar se a substring de S formada
2 // pelos elementos entre os índices i e j
3 // é um palindromo ou ão.
4
5 char s[MAX];
6 int calculado[MAX][MAX]; // inciado com false, ou 0
7 int tabela[MAX][MAX];
8
9 int is_palin(int i, int j){
10     if(calculado[i][j]){
11         return tabela[i][j];
12     }
13     if(i == j) return true;
14     if(i + 1 == j) return s[i] == s[j];
15
16     int ans = false;
17     if(s[i] == s[j]){
18         if(is_palin(i+1, j-1)){
19             ans = true;
20         }
21     }
22     calculado[i][j] = true;
23     tabela[i][j] = ans;
24     return ans;
25 }

```

### 7.4 Edit Distance

```

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

```

## 7.5 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 }

```

## 7.6 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 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 }

```

## 7.8 Kadane

```

1 // Description:
2 // Finds the maximum (or minimum) sum of some
    subarray of a given array
3
4 // Problem:
5 // https://leetcode.com/problems/maximum-subarray/
    description/
6
7 // Complexity:
8 // O(n)
9
10 // Notes
11 // To solve the minimum subarray problem, start the
    variable ans with INF and change the max
    operations to min operations
12 // To not count the empty subarray as a subarray,
    start the variable ans with -INF
13 // To get the biggest possible subarray with that sum
    , change if (curr > ans) to if (curr >= ans)
14 // If the empty subarray is the answer, start and end
    will be equal to -1
15
16 int ans = 0, curr = 0;
17 int startidx = 0, start = -1, end = -1;
18
19 for (int i = 0; i < n; i++) {
20     // MAXIMUM SUBARRAY PROBLEM
21     curr = max(curr + v[i], v[i]);
22     ans = max(ans, curr);
23
24     /*
25     RECOVER INDEXES MAXIMUM SUBARRAY PROBLEM
26     if (curr + v[i] < v[i]) {
27         startidx = i;
28         curr = v[i];
29     }
30     else curr += v[i];
31
32     if (curr > ans) {
33         ans = curr;
34         start = startidx;
35         end = i;
36     }
37     */
38
39     // MINIMUM SUBARRAY PROBLEM
40     // curr = min(curr + v[i], v[i]);
41     // ans = min(ans, curr);
42 }

```

```

43  /*
44  // MINIMUM SUBARRAY PROBLEM
45  if (curr + v[i] > v[i]) {
46      startidx = i;
47      curr = v[i];
48  }
49  else curr += v[i];
50
51  if (curr < ans) {
52      ans = curr;
53      start = startidx;
54      end = i;
55  }
56  */
57 }
58
59 // cout << ans << ' ' << start << ' ' << end << '\n';

```

## 7.9 Divide And Conquer

```

1  // Description:
2  // Divide the array in k intervals
3
4  // Problem:
5  // https://cses.fi/problemset/task/2086/
6
7  // Recurrence
8  // dp(i, j) = min(dp(i - 1, k - 1), C(k, j))
9
10 int dp[MAX][MAX];
11
12 vector<int> v, psum;
13 int k, n;
14
15 int C(int start, int end) {
16     if (start == 0) return psum[end] * psum[end];
17
18     return (psum[end] - psum[start - 1]) * (psum[end] -
19         psum[start - 1]);
20 }
21 void solve(int sz, int l, int r, int optl, int opttr)
22 {
23     if (l > r) return;
24
25     int mid = l + (r - l) / 2;
26
27     pii best = {INF, l};
28
29     for (int j = optl; j < min(mid, opttr + 1); j++) {
30         best = min(best, make_pair(C(j + 1, mid) + dp[sz
31             - 1][j], j));
32     }
33
34     dp[sz][mid] = best.first;
35
36     int opt = best.second;
37     solve(sz, l, mid - 1, optl, opt);
38     solve(sz, mid + 1, r, opt, opttr);
39 }
40
41 int32_t main() {
42     ios::sync_with_stdio(false);
43     cin.tie(NULL);
44
45     cin >> n >> k;
46     v.resize(n);
47     psum.assign(n, 0);
48
49     for (int idx = 0; idx < n; idx++) {
50         for (int sz = 0; sz <= n; sz++) {

```

```

51     }
52
53     for (int i = 0; i < n; i++) {
54         cin >> v[i];
55         psum[i] = v[i];
56         if (i != 0) psum[i] += psum[i - 1];
57     }
58
59     for (int i = 0; i < n; i++) {
60         dp[1][i] = C(0, i);
61     }
62
63     for (int sz = 2; sz <= n; sz++) {
64         solve(sz, 1, n, 0, n - 1);
65
66         /* for (int idx = 0; idx < n; idx++) {
67             for (int i = 0; i < idx; i++) {
68                 dp[sz][idx] = min(dp[sz][idx], dp[sz - 1][i]
69                     + C(i + 1, idx));
70             }
71         } */
72     }
73
74     cout << dp[k][n - 1] << '\n';
75
76     return 0;
77 }

```

## 7.10 Knuth

```

1  // Description
2  // Optimization of range dp
3
4  // Problem
5  // https://www.spoj.com/problems/BRKSTRNG/
6
7  // Recurrence
8  // dp(i, j) = min(dp(i, k) + d[(k + 1, j) + C(i, j)])
9     for (int k = i; k < j; k++)
10
11 int dp[MAX][MAX], opt[MAX][MAX];
12 vector<int> v;
13 int n, m;
14
15 int solve(int l, int r) {
16     if (abs(l - r) <= 1) return 0;
17
18     int& memo = dp[l][r];
19     if (memo != -1) return memo;
20
21     memo = INF;
22     for (int i = l + 1; i < r; i++) {
23         memo = min(memo, solve(l, i) + solve(i, r) + v[r]
24             - v[l]);
25     }
26
27     return memo;
28 }
29
30 int32_t main() {
31     ios::sync_with_stdio(false);
32     cin.tie(NULL);
33
34     while (cin >> n >> m) {
35         v.resize(m + 2, 0);
36         v[m + 1] = n;
37         for (int i = 1; i <= m; i++) {
38             cin >> v[i];
39         }
40
41         for (int i = 0; i <= m + 1; i++) {
42             for (int j = 0; j <= m + 1; j++) {

```

```

41     opt[i][i] = i;
42     dp[i][i] = 0;
43     dp[i][j] = INF;
44     if (abs(i - j) <= 1) dp[i][j] = 0;
45     if (abs(i - j) <= 1) opt[i][j] = i;
46 }
47 }
48
49 // memset(dp, -1, sizeof(dp));
50 // cout << solve(0, m + 1) << '\n';
51
52 for (int l = m; l >= 0; l--) {
53     for (int r = l + 1; r <= m + 1; r++) {
54         if (abs(l - r) <= 1) continue;
55         /* for (int i = l + 1; i < r; i++) {
56             dp[l][r] = min(dp[l][r], dp[l][i] + dp[i][r
57 ] + v[r] - v[l]);
58         } */
59
60         ll ans = INF;
61         for (int k=opt[l][r-1]; k<=min(r-1, opt[l
62 +1][r]); k++) {
63             ll best = dp[l][k] + dp[k][r];
64             if (ans > best) {
65                 ans = best;
66                 opt[l][r] = k;
67             }
68         }
69         dp[l][r] = ans + v[r] - v[l];
70     }
71 }
72 cout << dp[0][m + 1] << '\n';
73 }
74 return 0;
75 }

```

## 7.11 Cht

```

1 // Description:
2 // Write in terms of a line  $y = ax + b$ 
3
4 // Problem:
5 // https://atcoder.jp/contests/dp/tasks/dp_z/
6
7 int n, c;
8 vector<int> h;
9 int dp[MAX];
10
11 const ll is_query = -INF;
12 struct Line{
13     ll m, b;
14     mutable function<const Line*> succ;
15     bool operator<(const Line& rhs) const{
16         if(rhs.b != is_query) return m < rhs.m;
17         const Line* s = succ();
18         if(!s) return 0;
19         ll x = rhs.m;
20         return b - s->b < (s->m - m) * x;
21     }
22 };
23 struct Cht : public multiset<Line>{ // maintain max m
24     *x+b
25     bool bad(iterator y){
26         auto z = next(y);
27         if(y == begin()){
28             if(z == end()) return 0;
29             return y->m == z->m && y->b <= z->b;
30         }
31         auto x = prev(y);
32         if(z == end()) return y->m == x->m && y->b <=
33 x->b;

```

```

32         return (ld)(x->b - y->b)*(z->m - y->m) >= (ld
33 )(y->b - z->b)*(y->m - x->m);
34     }
35     void insert_line(ll m, ll b){ // min -> insert (-
36 m,-b) -> -eval()
37         m *= -1; b *= -1;
38         auto y = insert({ m, b });
39         y->succ = [=]{ return next(y) == end() ? 0 :
40 &*next(y); };
41         if(bad(y)){ erase(y); return; }
42         while(next(y) != end() && bad(next(y))) erase
43 (next(y));
44         while(y != begin() && bad(prev(y))) erase(
45 prev(y));
46     }
47     ll eval(ll x){
48         auto l = *lower_bound((Line) { x, is_query })
49 ;
50         return -(l.m * x + l.b);
51     }
52 };
53
54 int cost(int a, int b) {
55     return (h[b] - h[a]) * (h[b] - h[a]) + c;
56 }
57
58 int32_t main() {
59     ios::sync_with_stdio(false);
60     cin.tie(NULL);
61
62     cin >> n >> c;
63     h.resize(n);
64
65     for (int i = 0; i < n; i++) {
66         cin >> h[i];
67         dp[i] = INF;
68     }
69
70     dp[0] = 0;
71     Cht cht = Cht();
72     cht.insert_line(-2 * h[0], h[0] * h[0] + dp[0]);
73
74     for (int i = 1; i < n; i++) {
75         dp[i] = h[i] * h[i] + c + cht.eval(h[i]);
76
77         cht.insert_line(-2 * h[i], h[i] * h[i] + dp[i]);
78
79         // for (int j = 0; j < i; j++) {
80         //     dp[i] = min(dp[i], dp[j] + cost(j, i));
81         // }
82     }
83
84     cout << dp[n - 1] << '\n';
85
86     return 0;
87 }

```

## 8 Graphs

### 8.1 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 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     bool sptSet[V];
56     for (int i = 0; i < V; i++)
57         dist[i] = INT_MAX, sptSet[i] = false;
58
59     dist[src] = 0;
60
61     for (int count = 0; count < V - 1; count++) {
62         int u = minDistance(dist, sptSet, V);
63
64         sptSet[u] = true;
65
66         for (int v = 0; v < V; v++)
67             if (!sptSet[v] && adj[u][v]
68                 && dist[u] != INT_MAX
69                 && dist[u] + adj[u][v] < dist[v])
70                 dist[v] = dist[u] + adj[u][v];
71     }
72 }
73
74 }

```

## 8.2 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 }

```

## 8.3 Eulerian Undirected

```

1 // Description:
2 // Hierholzer's Algorithm
3 // An Eulerian path is a path that passes through
4 // every edge exactly once.
5 // An Eulerian circuit is an Eulerian path that
6 // starts and ends on the same node.
7
8 // An Eulerian path exists in an undirected graph if
9 // the degree of every node is even (not counting
10 // self-edges)
11 // except for possibly exactly two nodes that have
12 // and odd degree (start and end nodes).
13 // An Eulerian circuit exists in an undirected graph
14 // if the degree of every node is even.
15
16 // The graph has to be connected (except for isolated
17 // nodes which are allowed because there
18 // are no edges connected to them).
19
20 // Problem:
21 // https://cses.fi/problemset/task/1691
22
23 // Complexity:
24 // O(E * log(E)) where E is the number of edges
25
26 // How to use
27 // Check whether the path exists before trying to
28 // find it
29 // Find the root - any node that has at least 1
30 // outgoing edge
31 // (if the problem requires that you start from a
32 // node v, the root will be the node v)
33 // Count the degree;
34 //
35 // for (int i = 0; i < m; i++) {
36 //     int a, b; cin >> a >> b;
37 //     adj[a].pb(b); adj[b].pb(a);
38 //     root = a;
39 //     degree[a]++; degree[b]++;

```

```

30 // }
31
32 // Notes
33 // If you want to find a path start and ending nodes
   v and u
34 // if ((is_eulerian(n, root, start, end) != 1) || (
   start != v) || (end != u)) cout << "IMPOSSIBLE\n"
35
36 // It can be speed up to work on O(E) on average by
   using unordered_set instead of set
37
38 // It works when there are self loops, but not when
   there are multiple edges
39 // If the graph has multiple edges, add more notes to
   simulate the edges
40 // e.g
41 // 1 2
42 // 1 2
43 // 1 2
44 // becomes
45 // 3 4
46 // 4 1
47 // 1 2
48
49 vector<bool> visited;
50 vector<int> degree;
51 vector<vector<int>> adj;
52
53 void dfs(int v) {
54     visited[v] = true;
55     for (auto u : adj[v]) {
56         if (!visited[u]) dfs(u);
57     }
58 }
59
60 int is_eulerian(int n, int root, int& start, int& end
   ) {
61     start = -1, end = -1;
62     if (n == 1) return 2; // only one node
63     visited.assign(n + 1, false);
64     dfs(root);
65
66     for (int i = 1; i <= n; i++) {
67         if (!visited[i] && degree[i] > 0) return 0;
68     }
69
70     for (int i = 1; i <= n; i++) {
71         if (start == -1 && degree[i] % 2 == 1) start = i;
72         else if (end == -1 && degree[i] % 2 == 1) end = i;
73         ;
74         else if (degree[i] % 2 == 1) return 0;
75     }
76
77     if (start == -1 && end == -1) {start = root; end =
       root; return 2;} // has eulerian circuit and path
78     if (start != -1 && end != -1) return 1; // has
       eulerian path
79     return 0; // no eulerian path nor circuit
80 }
81
82 vector<int> path;
83 vector<set<int>> mark;
84
85 void dfs_path(int v) {
86     visited[v] = true;
87
88     while (degree[v] != 0) {
89         degree[v]--;
90         int u = adj[v][degree[v]];
91         if (mark[v].find(u) != mark[v].end()) continue;
92         mark[v].insert(u);
93         mark[u].insert(v);
94         int next_edge = adj[v][degree[v]];

```

```

94     dfs_path(next_edge);
95 }
96 path.pb(v);
97 }
98
99 void find_path(int n, int start) {
100     path.clear();
101     mark.resize(n + 1);
102     visited.assign(n + 1, false);
103     dfs_path(start);
104 }

```

## 8.4 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)
       , weight(weight) {}
46
47     bool operator<(const Edge& other) const {
48         return weight < other.weight;
49     }
50
51     bool operator>(const Edge& other) const {
52         return weight > other.weight;
53     }
54 };
55
56 vector<Edge> kruskal(vector<Edge> edges, int n) {
57     vector<Edge> result; // arestas da MST
58     long long cost = 0;

```

```

59     sort(edges.begin(), edges.end());
60
61     DSU dsu(n);
62
63     for (auto e : edges) {
64         if (!dsu.same(e.u, e.v)) {
65             cost += e.weight;
66             result.push_back(e);
67             dsu.unite(e.u, e.v);
68         }
69     }
70
71     return result;
72 }
73 }

```

## 8.5 Negative Cycle

```

1 // Description
2 // Detects any cycle in which the sum of edge weights
  is negative.
3 // Alternatively, we can detect whether there is a
  negative cycle
4 // starting from a specific vertex.
5
6 // Problem:
7 // https://cses.fi/problemset/task/1197
8
9 // Complexity:
10 // O(n * m)
11
12 // Notes
13 // In order to consider only the negative cycles
  located on the path from a to b,
14 // Reverse the graph, run a dfs from node b and mark
  the visited nodes
15 // Consider only the edges that connect to visited
  nodes when running bellman-ford
16 // on the normal graph
17
18 struct Edge {
19     int a, b, cost;
20     Edge(int a, int b, int cost) : a(a), b(b), cost(
  cost) {}
21 };
22
23 int n, m;
24 vector<Edge> edges;
25 const int INF = 1e9+10;
26
27 void negative_cycle() {
28     // uncomment to find negative cycle starting from a
  vertex v
29     // vector<int> d(n + 1, INF);
30     // d[v] = 0;
31     vector<int> d(n + 1, 0);
32     vector<int> p(n + 1, -1);
33     int x;
34     // uncomment to find all negative cycles
35     // // set<int> s;
36     for (int i = 1; i <= n; ++i) {
37         x = -1;
38         for (Edge e : edges) {
39             // if (d[e.a] >= INF) continue;
40             if (d[e.b] > d[e.a] + e.cost) {
41                 // d[e.b] = max(-INF, d[e.a] + e.cost);
42                 d[e.b] = d[e.a] + e.cost;
43                 p[e.b] = e.a;
44                 x = e.b;
45                 // // s.insert(e.b);
46             }
47         }
48     }

```

```

49     if (x == -1)
50         cout << "NO\n";
51     else {
52         // // int y = all nodes in set s
53         int y = x;
54         for (int i = 1; i <= n; ++i) {
55             y = p[y];
56         }
57
58         vector<int> path;
59         for (int cur = y; cur = p[cur]) {
60             path.push_back(cur);
61             if (cur == y && path.size() > 1) break;
62         }
63         reverse(path.begin(), path.end());
64
65         cout << "YES\n";
66         for (int u : path)
67             cout << u << ' ';
68         cout << '\n';
69     }
70 }
71 }

```

## 8.6 Floyd Warshall

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using ll = long long;
5
6 const int MAX = 507;
7 const long long INF = 0x3f3f3f3f3f3f3f3fLL;
8
9 ll dist[MAX][MAX];
10 int n;
11
12 void floyd_warshall() {
13     for (int i = 0; i < n; i++) {
14         for (int j = 0; j < n; j++) {
15             if (i == j) dist[i][j] = 0;
16             else if (!dist[i][j]) dist[i][j] = INF;
17         }
18     }
19
20     for (int k = 0; k < n; k++) {
21         for (int i = 0; i < n; i++) {
22             for (int j = 0; j < n; j++) {
23                 // trata o caso no qual o grafo tem
  arestas com peso negativo
24                 if (dist[i][k] < INF && dist[k][j] <
  INF){
25                     dist[i][j] = min(dist[i][j], dist
  [i][k] + dist[k][j]);
26                 }
27             }
28         }
29     }
30 }

```

## 8.7 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 }

```

## 8.8 Eulerian Directed

```

1 // Description:
2 // Hierholzer's Algorithm
3 // An Eulerian path is a path that passes through
   every edge exactly once.
4 // An Eulerian circuit is an Eulerian path that
   starts and ends on the same node.
5
6 // An Eulerian path exists in an directed graph if
   the indegree and outdegree is equal
7 // for every node (not counting self-edges)
8 // except for possibly exactly one node that have
   outdegree - indegree = 1
9 // and one node that has indegree - outdegree = 1 (
   start and end nodes).
10 // An Eulerian circuit exists in an directed graph if
   the indegree and outdegree is equal for every
   node.
11
12 // The graph has to be conected (except for isolated
   nodes which are allowed because there
13 // are no edges connected to them).
14
15 // Problem:
16 // https://cses.fi/problemset/task/1693
17
18 // Complexity:
19 // O(E) where E is the number of edges
20

```

```

21 // How to use
22 // Check whether the path exists before trying to
   find it
23 // Find the root - any node that has at least 1
   outgoing edge
24 // (if the problem requires that you start from a
   node v, the root will be the node v)
25 // Count the degree;
26 //
27 // for (int i = 0; i < m; i++) {
28 //     int a, b; cin >> a >> b;
29 //     adj[a].pb(b);
30 //     root = a;
31 //     outdegree[a]++; indegree[b]++;
32 // }
33
34 // Notes
35 // It works when there are self loops, but not when
   there are multiple edges
36
37 vector<bool> visited;
38 vector<int> outdegree, indegree;
39 vector<vector<int>> adj, undir;
40
41 void dfs(int v) {
42     visited[v] = true;
43     for (auto u : undir[v]) {
44         if (!visited[u]) dfs(u);
45     }
46 }
47
48 int is_eulerian(int n, int root, int &start, int& end
) {
49     start = -1, end = -1;
50     if (n == 1) return 2; // only one node
51     visited.assign(n + 1, false);
52     dfs(root);
53
54     for (int i = 1; i <= n; i++) {
55         if (!visited[i] && (i == n || i == 1 || outdegree
[i] + indegree[i] > 0)) return 0;
56     }
57
58     // start => node with indegree - outdegree = 1
59     // end => node with outdegree - indegree = 1
60     for (int i = 1; i <= n; i++) {
61         if (start == -1 && indegree[i] - outdegree[i] ==
1) start = i;
62         else if (end == -1 && outdegree[i] - indegree[i]
== 1) end = i;
63         else if (indegree[i] != outdegree[i]) return 0;
64     }
65
66     if (start == -1 && end == -1) {start = root; end =
root; return 2;} // has eulerian circuit and path
67     if (start != -1 && end != -1) {swap(start, end);
return 1;} // has eulerian path
68     return 0; // no eulerian path nor circuit
69 }
70
71 vector<int> path;
72
73 void dfs_path(int v) {
74     visited[v] = true;
75
76     while (outdegree[v] != 0) {
77         int u = adj[v][--outdegree[v]];
78         int next_edge = adj[v][outdegree[v]];
79         dfs_path(next_edge);
80     }
81     path.pb(v);
82 }
83

```

```

84 void find_path(int n, int start) {
85     path.clear();
86     visited.assign(n + 1, false);
87     dfs_path(start);
88     reverse(path.begin(), path.end());
89 }

```

## 8.9 Lca

```

1 // Description:
2 // Find the lowest common ancestor between two nodes
  in a tree
3
4 // Problem:
5 // https://cses.fi/problemset/task/1135
6
7 // Complexity:
8 // O(log n)
9
10 // How to use:
11 // preprocess();
12 // lca(a, b);
13
14 // Notes
15 // To calculate the distance between two nodes use
  the following formula
16 // level_peso[a] + level_peso[b] - 2*level_peso[lca(a
  , b)]
17
18 // If you just need to know if a node is the ancestor
  of another node or not
19
20 vector<vector<int>> adj;
21 vector<int> tin, tout;
22
23 void dfs(int v, int p, int& idx) {
24     tin[v] = idx++;
25
26     for (auto u : adj[v]) {
27         if (u == p) continue;
28         dfs(u, v, idx);
29     }
30
31     tout[v] = idx++;
32 }
33
34 bool is_ancestor(int a, int b) {
35     return (tin[a] >= tin[b] && tout[b] <= tout[a])
36     || (tin[b] >= tin[a] && tout[a] <= tout[b]);
37 }
38
39 // LCA
40
41 const int MAX = 2e5+10;
42 const int BITS = 30;
43
44 vector<pii> adj[MAX];
45 vector<bool> visited(MAX);
46
47 int up[MAX][BITS + 1];
48 int level[MAX];
49 int level_peso[MAX];
50
51 void find_level() {
52     queue<pii> q;
53
54     q.push(mp(1, 0));
55     visited[1] = true;
56
57     while (!q.empty()) {
58         auto [v, depth] = q.front();
59         q.pop();
60         level[v] = depth;

```

```

61     for (auto [u,d] : adj[v]) {
62         if (!visited[u]) {
63             visited[u] = true;
64             up[u][0] = v;
65             q.push(mp(u, depth + 1));
66         }
67     }
68 }
69 }
70 }
71
72 void find_level_peso() {
73     queue<pii> q;
74
75     q.push(mp(1, 0));
76     visited[1] = true;
77
78     while (!q.empty()) {
79         auto [v, depth] = q.front();
80         q.pop();
81         level_peso[v] = depth;
82
83         for (auto [u,d] : adj[v]) {
84             if (!visited[u]) {
85                 visited[u] = true;
86                 up[u][0] = v;
87                 q.push(mp(u, depth + d));
88             }
89         }
90     }
91 }
92
93 int lca(int a, int b) {
94     // get the nodes to the same level
95     int mn = min(level[a], level[b]);
96
97     for (int j = 0; j <= BITS; j++) {
98         if (a != -1 && ((level[a] - mn) & (1 << j))) a
99         = up[a][j];
100         if (b != -1 && ((level[b] - mn) & (1 << j))) b
101         = up[b][j];
102     }
103
104     // special case
105     if (a == b) return a;
106
107     // binary search
108     for (int j = BITS; j >= 0; j--) {
109         if (up[a][j] != up[b][j]) {
110             a = up[a][j];
111             b = up[b][j];
112         }
113     }
114     return up[a][0];
115 }
116
117 void preprocess() {
118     visited = vector<bool>(MAX, false);
119     find_level();
120     visited = vector<bool>(MAX, false);
121     find_level_peso();
122
123     for (int j = 1; j <= BITS; j++) {
124         for (int i = 1; i <= n; i++) {
125             if (up[i][j - 1] != -1) up[i][j] = up[up[i][j - 1]][j - 1];
126         }
127     }
128 }

```

## 8.10 Ford Fulkerson Edmonds Karp

```

1 // Description:

```

```

2 // Obtains the maximum possible flow rate given a
  network. A network is a graph with a single
  source vertex and a single sink vertex in which
  each edge has a capacity
3
4 // Complexity:
5 //  $O(V * E^2)$  where V is the number of vertex and E
  is the number of edges
6
7 int n;
8 vector<vector<int>> capacity;
9 vector<vector<int>> adj;
10
11 int bfs(int s, int t, vector<int>& parent) {
12     fill(parent.begin(), parent.end(), -1);
13     parent[s] = -2;
14     queue<pair<int, int>> q;
15     q.push({s, INF});
16
17     while (!q.empty()) {
18         int cur = q.front().first;
19         int flow = q.front().second;
20         q.pop();
21
22         for (int next : adj[cur]) {
23             if (parent[next] == -1 && capacity[cur][
next]) {
24                 parent[next] = cur;
25                 int new_flow = min(flow, capacity[cur
][next]);
26                 if (next == t)
27                     return new_flow;
28                 q.push({next, new_flow});
29             }
30         }
31     }
32
33     return 0;
34 }
35
36 int maxflow(int s, int t) {
37     int flow = 0;
38     vector<int> parent(n);
39     int new_flow;
40
41     while (new_flow = bfs(s, t, parent)) {
42         flow += new_flow;
43         int cur = t;
44         while (cur != s) {
45             int prev = parent[cur];
46             capacity[prev][cur] -= new_flow;
47             capacity[cur][prev] += new_flow;
48             cur = prev;
49         }
50     }
51
52     return flow;
53 }

```

## 8.11 Kuhn

```

1 // Description
2 // Matching algorithm for unweighted bipartite graph
  ::
3
4 // Problem:
5 // https://codeforces.com/gym/104252/problem/H
6
7 // Complexity:
8 //  $O(V * E)$  in which V is the number of vertexes and
  E is the number of edges
9
10 // Notes:

```

```

11 // Indexed at zero
12
13 int n, k;
14 // adjacency list
15 vector<vector<int>> g;
16 vector<int> mt;
17 vector<bool> used;
18
19 bool try_kuhn(int v) {
20     if (used[v])
21         return false;
22     used[v] = true;
23     for (int to : g[v]) {
24         if (mt[to] == -1 || try_kuhn(mt[to])) {
25             mt[to] = v;
26             return true;
27         }
28     }
29     return false;
30 }
31
32 int main() {
33     // ... reading the graph g ...
34
35     mt.assign(k, -1);
36     vector<bool> used1(n, false);
37     for (int v = 0; v < n; ++v) {
38         for (int to : g[v]) {
39             if (mt[to] == -1) {
40                 mt[to] = v;
41                 used1[v] = true;
42                 break;
43             }
44         }
45     }
46     for (int v = 0; v < n; ++v) {
47         if (used1[v])
48             continue;
49         used.assign(n, false);
50         try_kuhn(v);
51     }
52
53     for (int i = 0; i < k; ++i)
54         if (mt[i] != -1)
55             printf("%d %d\n", mt[i] + 1, i + 1);
56 }

```

## 8.12 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     }

```

### 8.13 Hld Vertex

```

1 // Description:
2 // Make queries and updates between two vertexes on a
3 // tree
4 // Query path - query path (a, b) inclusive
5 // Update path - update path (a, b) inclusive
6 // Query subtree - query subtree of a
7 // Update subtree - update subtree of a
8 // Update - update vertex or edge
9 // Lca - get lowest common ancestor of a and b
10 // Search - perform a binary search to find the last
11 // node with a certain property
12 // on the path from a to the root
13
14 // Problem:
15 // https://codeforces.com/gym/101908/problem/L
16
17 // Complexity:
18 // O(log ^2 n) for both query and update
19
20 // How to use:
21 // HLD hld = HLD(n + 1, adj)
22
23 // Notes
24 // Change the root of the tree on the constructor if
25 // it's different from 1
26 // Use together with Segtree
27
28 typedef long long ftype;
29
30 struct HLD {
31     vector<int> parent;
32     vector<int> pos;
33     vector<int> head;
34     vector<int> subtree_size;
35     vector<int> level;
36     vector<int> heavy_child;
37     vector<ftype> subtree_weight;
38     vector<ftype> path_weight;
39     vector<vector<int>> adj;
40     vector<int> at;
41     Segtree seg = Segtree(0);
42     int cpos;
43     int n;
44     int root;

```

```

45     vector<vector<int>> up;
46
47     HLD() {}
48
49     HLD(int n, vector<vector<int>>& adj, int root = 1)
50     : adj(adj), n(n), root(root) {
51         seg = Segtree(n);
52         cpos = 0;
53         at.resize(n);
54         parent.resize(n);
55         pos.resize(n);
56         head.resize(n);
57         subtree_size.assign(n, 1);
58         level.assign(n, 0);
59         heavy_child.assign(n, -1);
60         parent[root] = -1;
61         dfs(root, -1);
62         decompose(root, -1);
63     }
64
65     void dfs(int v, int p) {
66         parent[v] = p;
67         if (p != -1) level[v] = level[p] + 1;
68         for (auto u : adj[v]) {
69             if (u != p) {
70                 dfs(u, v);
71                 subtree_size[v] += subtree_size[u];
72                 if (heavy_child[v] == -1 || subtree_size[u] >
73                     subtree_size[heavy_child[v]]) heavy_child[v] = u
74             }
75         }
76     }
77
78     void decompose(int v, int chead) {
79         // start a new path
80         if (chead == -1) chead = v;
81
82         // consecutive ids in the hld path
83         at[cpos] = v;
84         pos[v] = cpos++;
85         head[v] = chead;
86
87         // if not a leaf
88         if (heavy_child[v] != -1) decompose(heavy_child[v], chead);
89
90         // light child
91         for (auto u : adj[v]){
92             // start new path
93             if (u != parent[v] && u != heavy_child[v])
94                 decompose(u, -1);
95         }
96     }
97
98     ftype query_path(int a, int b) {
99         if(pos[a] < pos[b]) swap(a, b);
100
101         if(head[a] == head[b]) return seg.query(pos[b], pos[a]);
102         return seg.f(seg.query(pos[head[a]], pos[a]), query_path(parent[head[a]], b));
103     }
104
105     // iterative
106     /*ftype query_path(int a, int b) {
107         ftype ans = 0;
108
109         while (head[a] != head[b]) {
110             if (level[head[a]] > level[head[b]]) swap(a, b);
111             ans = seg.merge(ans, seg.query(pos[head[b]], pos[b]));
112         }
113         return seg.query(pos[a], pos[b]);
114     }

```



```

106     b = parent[head[b]];
107 }
108
109 if (level[a] > level[b]) swap(a, b);
110 ans = seg.merge(ans, seg.query(pos[a], pos[b]));
111 return ans;
112 }*/
113
114 ftype query_subtree(int a) {
115     return seg.query(pos[a], pos[a] + subtree_size[a]
116         - 1);
117 }
118
119 void update_path(int a, int b, int x) {
120     if(pos[a] < pos[b]) swap(a, b);
121
122     if(head[a] == head[b]) return (void)seg.update(
123         pos[b], pos[a], x);
124     seg.update(pos[head[a]], pos[a], x); update_path(
125         parent[head[a]], b, x);
126 }
127
128 void update_subtree(int a, int val) {
129     seg.update(pos[a], pos[a] + subtree_size[a] - 1,
130         val);
131 }
132
133 void update(int a, int val) {
134     seg.update(pos[a], pos[a], val);
135 }
136
137 //edge
138 void update(int a, int b, int val) {
139     if (level[a] > level[b]) swap(a, b);
140     update(b, val);
141 }
142
143 int lca(int a, int b) {
144     if(pos[a] < pos[b]) swap(a, b);
145     return head[a] == head[b] ? b : lca(parent[head[a]], b);
146 }
147
148 void search(int a) {
149     a = parent[a];
150     if (a == -1) return;
151     if (seg.query(pos[head[a]], pos[head[a]] +
152         subtree_size[head[a]] - 1) + pos[a] - pos[head[a]] + 1
153         == subtree_size[head[a]]) {
154         seg.update(pos[head[a]], pos[a], 1);
155         return search(parent[head[a]]);
156     }
157     int l = pos[head[a]], r = pos[a] + 1;
158     while (l < r) {
159         int m = (l+r)/2;
160         if (seg.query(m, m+subtree_size[at[m]] - 1) + pos
161             [a] - m + 1 == subtree_size[at[m]]) {
162             r = m;
163         }
164         else l = m+1;
165     }
166     seg.update(l, pos[a], 1);
167 }
168
169 /* k-th ancestor of x
170 int x, k; cin >> x >> k;
171
172 for (int b = 0; b <= BITS; b++) {
173     if (x != -1 && (k & (1 << b))) {
174         x = up[x][b];
175     }
176 }
177
178 cout << x << '\n';
179 */
180 void preprocess() {
181     up.assign(n + 1, vector<int>(31, -1));
182
183     for (int i = 1; i < n; i++) {
184         up[i][0] = parent[i];
185     }
186
187     for (int i = 1; i < n; i++) {
188         for (int j = 1; j <= 30; j++) {
189             if (up[i][j - 1] != -1) up[i][j] = up[up[i][j - 1]][j - 1];
190         }
191     }
192 }
193
194 int getKth(int p, int q, int k) {
195     int a = lca(p, q), d;
196
197     if( a == p ) {
198         d = level[q] - level[p] + 1;
199         swap(p, q);
200         k = d - k + 1;
201     }
202     else if( a == q ) ;
203     else {
204         if( k > level[p] - level[a] + 1 ) {
205             d = level[p] + level[q] - 2 * level[a] +
206             1;
207             k = d - k + 1;
208             swap(p, q);
209         }
210         else ;
211     }
212     int lg; for( lg = 1; (1 << lg) <= level[p]; ++
213         lg ); lg--;
214     k--;
215     for( int i = lg; i >= 0; i-- ) {
216         if( (1 << i) <= k ) {
217             p = up[p][i];
218             k -= ( 1 << i );
219         }
220     }
221     return p;
222 }
223
224 };
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

## 8.14 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            }
25        }
26    }
27 }
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```



```

21         else if(colors[curr][item]+vzs ==
22             vmax[curr]){
23             sum_num[curr] += item;
24         }
25         colors[curr][item] += vzs;
26     }
27 }
28 }
29 }
30 }
31
32 int32_t main() {
33     int n; cin >> n;
34
35     for (int i = 1; i <= n; i++) {
36         int a; cin >> a;
37         colors[i][a] = 1;
38         vmax[i] = 1;
39         sum_num[i] = a;
40     }
41
42     for (int i = 1; i < n; i++) {
43         int a, b; cin >> a >> b;
44
45         adj[a].push_back(b);
46         adj[b].push_back(a);
47     }
48
49     process_colors(1, 0);
50
51     for (int i = 1; i <= n; i++) {
52         cout << sum_num[i] << (i < n ? " " : "\n");
53     }
54
55     return 0;
56 }
57
58 }
59
60

```

## 8.15 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 }

```

## 8.16 Min Cost Max Flow

```

1  // Dinitz Min Cost
2  const int INF = 0x3f3f3f3f3f3f3f3f;
3
4  struct Dinitz {
5      struct Edge {
6          int v, u, cap, flow=0, cost;
7          Edge(int v, int u, int cap, int cost) : v(v), u(u), cap(cap), cost(cost) {}
8      };
9
10     int n, s, t;
11     Dinitz(int n, int s, int t) : n(n), s(s), t(t) {
12         adj.resize(n);
13     }
14

```

```

15 vector<Edge> edges;
16 vector<vector<int>> adj;
17 void add_edge(int v, int u, int cap, int cost) {
18     edges.emplace_back(v, u, cap, cost);
19     adj[v].push_back(size(edges)-1);
20     edges.emplace_back(u, v, 0, -cost);
21     adj[u].push_back(size(edges)-1);
22 }
23
24 vector<int> dist;
25 bool spfa() {
26     dist.assign(n, INF);
27
28     queue<int> Q;
29     vector<bool> inqueue(n, false);
30
31     dist[s] = 0;
32     Q.push(s);
33     inqueue[s] = true;
34
35     vector<int> cnt(n);
36
37     while (!Q.empty()) {
38         int v = Q.front(); Q.pop();
39         inqueue[v] = false;
40
41         for (auto eid : adj[v]) {
42             auto const& e = edges[eid];
43             if (e.cap - e.flow <= 0) continue;
44             if (dist[e.u] > dist[e.v] + e.cost) {
45                 dist[e.u] = dist[e.v] + e.cost;
46                 if (!inqueue[e.u]) {
47                     Q.push(e.u);
48                     inqueue[e.u] = true;
49                 }
50             }
51         }
52     }
53
54     return dist[t] != INF;
55 }
56
57 int cost = 0;
58 vector<int> ptr;
59 int dfs(int v, int f) {
60     if (v == t || f == 0) return f;
61     for (auto &cid = ptr[v]; cid < size(adj[v]);) {
62         auto eid = adj[v][cid];
63         auto &e = edges[eid];
64         cid++;
65         if (e.cap - e.flow <= 0) continue;
66         if (dist[e.v] + e.cost != dist[e.u]) continue;
67         int newf = dfs(e.u, min(f, e.cap - e.flow));
68         if (newf == 0) continue;
69         e.flow += newf;
70         edges[eid^1].flow -= newf;
71         cost += e.cost * newf;
72         return newf;
73     }
74     return 0;
75 }
76
77 int total_flow = 0;
78 int flow() {
79     while (spfa()) {
80         ptr.assign(n, 0);
81         while (int newf = dfs(s, INF))
82             total_flow += newf;
83     }
84     return total_flow;
85 }
86 };
87 //}}}

```

## 8.17 Hungarian

```

1 // Description:
2 // A matching algorithm for weighted bipartite graphs
   that returns
3 // a perfect match
4
5 // Problem:
6 // https://codeforces.com/gym/103640/problem/H
7
8 // Complexity:
9 //  $O(V^3)$  in which V is the number of vertexs
10
11 // Notes:
12 // Indexed at 1
13
14 // n is the number of items on the right side and m
   the number of items
15 // on the left side of the graph
16
17 // Returns minimum assignment cost and which items
   were matched
18
19 pair<int, vector<pii>> hungarian(int n, int m, vector
   <vector<int>> A) {
20     vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
21     for (int i=1; i<=n; ++i) {
22         p[0] = i;
23         int j0 = 0;
24         vector<int> minv (m+1, INF);
25         vector<char> used (m+1, false);
26         do {
27             used[j0] = true;
28             int i0 = p[j0], delta = INF, j1;
29             for (int j=1; j<=m; ++j)
30                 if (!used[j]) {
31                     int cur = A[i0][j]-u[i0]-v[j];
32                     if (cur < minv[j])
33                         minv[j] = cur, way[j] = j0;
34                     if (minv[j] < delta)
35                         delta = minv[j], j1 = j;
36                 }
37             for (int j=0; j<=m; ++j)
38                 if (used[j])
39                     u[p[j]] += delta, v[j] -= delta;
40             else
41                 minv[j] -= delta;
42             j0 = j1;
43         } while (p[j0] != 0);
44         do {
45             int j1 = way[j0];
46             p[j0] = p[j1];
47             j0 = j1;
48         } while (j0);
49     }
50
51     vector<pair<int, int>> result;
52     for (int i = 1; i <= m; ++i){
53         result.push_back(make_pair(p[i], i));
54     }
55
56     int C = -v[0];
57
58     return mp(C, result);
59 }

```

## 8.18 2sat

```

1 // Description:
2 // Solves expression of the type  $(a \vee b) \wedge (c \vee d) \wedge$ 
    $(e \vee f)$ 
3
4 // Problem:

```

```

5 // https://cses.fi/problemset/task/1684
6
7 // Complexity:
8 // O(n + m) where n is the number of variables and m
  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
101     void topological_order(int v) {
102         visited[v] = true;
103
104         for (auto u : condensed[v]) {
105             if (!visited[u]) topological_order(u);
106         }
107
108         order.pb(v);
109     }
110
111     bool is_possible() {
112         component = 0;
113         for (int i = 1; i <= 2 * nodes; i++) {
114             if (!visited[i]) departure_time(i);
115         }
116
117         sort(departure.begin(), departure.end(),
118 greater<pii>());
119
120         visited.assign(2 * nodes + 1, false);
121
122         for (auto [_, node] : departure) {
123             if (!visited[node]) find_component(node,
124 ++component);
125         }
126
127         for (int i = 1; i <= nodes; i++) {
128             if (scc[i] == scc[i + nodes]) return
129 false;
130         }
131
132         return true;
133     }
134
135     int find_value(int e, vector<int> &ans) {
136         if (e > nodes && ans[e - nodes] != 2) return
137 !ans[e - nodes];
138         if (e <= nodes && ans[e + nodes] != 2) return
139 !ans[e + nodes];
140         return 0;
141     }
142
143     vector<int> find_ans() {
144         condensed.resize(component + 1);
145
146         for (int i = 1; i <= 2 * nodes; i++) {
147             for (auto u : adj[i]) {
148                 if (scc[i] != scc[u]) condensed[scc[i]
149 ]].pb(scc[u]);

```

```

143     }
144 }
145
146 visited.assign(component + 1, false);
147
148 for (int i = 1; i <= component; i++) {
149     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         if (op1 == '-' && op2 == '-') sat.add_or(sat.get_not(a), sat.get_not(b));
188         if (op1 == '+' && op2 == '-') sat.add_or(a, sat.get_not(b));
189         if (op1 == '-' && op2 == '+') sat.add_or(sat.get_not(a), b);
190     }
191
192     if (!sat.is_possible()) cout << "IMPOSSIBLE\n";
193     else {
194         vector<int> ans = sat.find_ans();
195         for (int i = 1; i <= m; i++) {
196             cout << (ans[i] == 1 ? '+' : '-') << ' ';
197         }
198         cout << '\n';
199     }
200
201     return 0;
202 }

```

## 8.19 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 // Problem:

```

```

6 // https://cses.fi/problemset/task/2177/
7
8 // Complexity:
9 // O(V + E) where V is the number of vertices and E
10 // is the number of edges
11
12 int n;
13 vector<vector<int>> adj;
14
15 vector<bool> visited;
16 vector<int> tin, low;
17 int timer;
18
19 void dfs(int v, int p) {
20     visited[v] = true;
21     tin[v] = low[v] = timer++;
22     for (int to : adj[v]) {
23         if (to == p) continue;
24         if (visited[to]) {
25             low[v] = min(low[v], tin[to]);
26         } else {
27             dfs(to, v);
28             low[v] = min(low[v], low[to]);
29             if (low[to] > tin[v]) {
30                 IS_BRIDGE(v, to);
31             }
32         }
33     }
34 }
35
36 void find_bridges() {
37     timer = 0;
38     visited.assign(n, false);
39     tin.assign(n, -1);
40     low.assign(n, -1);
41     for (int i = 0; i < n; ++i) {
42         if (!visited[i])
43             dfs(i, -1);
44     }
45 }

```

## 8.20 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 }

```

## 8.21 Prim

```

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

```

## 8.22 Blossom

```

1  // Description:
2  // Matching algorithm for general graphs (non-
   bipartite)
3
4  // Problem:
5  // https://acm.timus.ru/problem.aspx?space=1&num=1099
6
7  // Complexity:
8  // O(n^3)
9
10 // vector<pii> Blossom(vector<vector<int>>& graph) {
11 vector<int> Blossom(vector<vector<int>>& graph) {
12     int n = graph.size(), timer = -1;
13     vector<int> mate(n, -1), label(n), parent(n),
14         orig(n), aux(n, -1), q;
15     auto lca = [&](int x, int y) {
16         for (timer++; ; swap(x, y)) {
17             if (x == -1) continue;
18             if (aux[x] == timer) return x;

```

```

19         aux[x] = timer;
20         x = (mate[x] == -1 ? -1 : orig[parent[mate[x]
]]]);
21     }
22 };
23 auto blossom = [&](int v, int w, int a) {
24     while (orig[v] != a) {
25         parent[v] = w; w = mate[v];
26         if (label[w] == 1) label[w] = 0, q.push_back(w)
;
27         orig[v] = orig[w] = a; v = parent[w];
28     }
29 };
30 auto augment = [&](int v) {
31     while (v != -1) {
32         int pv = parent[v], nv = mate[pv];
33         mate[v] = pv; mate[pv] = v; v = nv;
34     }
35 };
36 auto bfs = [&](int root) {
37     fill(label.begin(), label.end(), -1);
38     iota(orig.begin(), orig.end(), 0);
39     q.clear();
40     label[root] = 0; q.push_back(root);
41     for (int i = 0; i < (int)q.size(); ++i) {
42         int v = q[i];
43         for (auto x : graph[v]) {
44             if (label[x] == -1) {
45                 label[x] = 1; parent[x] = v;
46                 if (mate[x] == -1)
47                     return augment(x), 1;
48                 label[mate[x]] = 0; q.push_back(mate[x]);
49             } else if (label[x] == 0 && orig[v] != orig[x]
) {
50                 int a = lca(orig[v], orig[x]);
51                 blossom(x, v, a); blossom(v, x, a);
52             }
53         }
54     }
55     return 0;
56 };
57 // Time halves if you start with (any) maximal
   matching.
58 for (int i = 0; i < n; i++)
59     if (mate[i] == -1)
60         bfs(i);
61 return mate;
62
63 /*
64 vector<bool> used(n, false);
65 vector<pii> ans;
66 for (int i = 0; i < n; i++) {
67     if (matching[i] == -1 || used[i]) continue;
68     used[i] = true;
69     used[matching[i]] = true;
70     ans.emplace_back(i, matching[i]);
71 }
72 return ans;
73 */

```

## 8.23 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         level[source] = 0;
103
104         while (!q.empty()) {
105             int node = q.front();
106             q.pop();
107
108             for (auto e : adj[node]) {
109                 if (level[e->to] == -1 && e->
   get_capacity() > 0) {
110                     level[e->to] = level[e->from] +
   1;
111                     q.push(e->to);
112                 }
113             }
114         }
115
116         return level[sink] != -1;
117     }
118
119     ll dfs(int v, ll flow) {
120         if (v == sink)
121             return flow;
122
123         int sz = adj[v].size();
124         for (int i = next[v]; i < sz; i++) {
125             Edge* e = adj[v][i];
126             if (level[e->to] == level[e->from] + 1 &&
   e->get_capacity() > 0) {
127                 ll bottleneck = dfs(e->to, min(flow,
   e->get_capacity()));
128                 if (bottleneck > 0) {
129                     e->augment(bottleneck);
130                     return bottleneck;
131                 }
132             }
133             next[v] = i + 1;
134         }
135         return 0;
136     }
137
138     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()
158 > 0) {
159             reach.pb(e->to);
160             visited[e->to] = true;
161             reachable(e->to);
162         }
163     }
164 }
165
166 void print_min_cut() {
167     reach.clear();
168     visited.assign(nodes + 1, false);
169     reach.pb(source);
170     reachable(source);
171
172     for (auto v : reach) {
173         for (auto e : adj[v]) {
174             if (!visited[e->to] && e->
175 get_capacity() == 0) {
176                 cout << e->from << ' ' << e->to
177 << '\n';
178             }
179         }
180     }
181
182 ll build_path(int v, int id, ll flow) {
183     visited[v] = true;
184     if (v == sink) {
185         return flow;
186     }
187
188     for (auto e : adj[v]) {
189         if (!visited[e->to] && e->get_flow() > 0)
190 {
191             visited[e->to] = true;
192             ll bottleneck = build_path(e->to, id,
193 min(flow, e->get_flow()));
194             if (bottleneck > 0) {
195                 path[id].pb(e->to);
196                 e->reverse(bottleneck);
197                 return bottleneck;
198             }
199         }
200     }
201
202     return 0;
203 }
204
205 void print_flow_path() {
206     path.clear();
207     ll sent = -1;
208     int id = -1;
209     while (sent != 0) {
210         visited.assign(nodes + 1, false);
211         path.pb(vector<int>{});
212         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 }

```

## 8.24 Bellman Ford

```

1 // Description:
2 // Finds the shortest path from a vertex v to any
3 // other vertex
4
5 // Problem:
6 // https://cses.fi/problemset/task/1673
7
8 // Complexity:
9 // O(n * m)
10
11 struct Edge {
12     int a, b, cost;
13     Edge(int a, int b, int cost) : a(a), b(b), cost(
14 cost) {}
15 };
16
17 int n, m;
18 vector<Edge> edges;
19 const int INF = 1e9+10;
20
21 void bellman_ford(int v, int t) {
22     vector<int> d(n + 1, INF);
23     d[v] = 0;
24     vector<int> p(n + 1, -1);
25
26     for (;;) {
27         bool any = false;
28         for (Edge e : edges) {
29             if (d[e.a] >= INF) continue;
30             if (d[e.b] > d[e.a] + e.cost) {
31                 d[e.b] = d[e.a] + e.cost;
32                 p[e.b] = e.a;
33                 any = true;
34             }
35         }
36         if (!any) break;

```

```

35 }
36
37 if (d[t] == INF)
38     cout << "No path from " << v << " to " << t << ".
39 ";
40 else {
41     vector<int> path;
42     for (int cur = t; cur != -1; cur = p[cur]) {
43         path.push_back(cur);
44     }
45     reverse(path.begin(), path.end());
46
47     cout << "Path from " << v << " to " << t << ": ";
48     for (int u : path) {
49         cout << u << ' ';
50     }
51 }

```

## 8.25 Hld Edge

```

1 // Description:
2 // Make queries and updates between two vertexes on a
3 tree
4 // Problem:
5 // https://www.spoj.com/problems/QTREE/
6 // Complexity:
7 // O(log ^2 n) for both query and update
8 // How to use:
9 // HLD hld = HLD(n + 1, adj)
10 // Notes
11 // Change the root of the tree on the constructor if
12 it's different from 1
13 // Use together with Segtree
14
15 struct HLD {
16     vector<int> parent;
17     vector<int> pos;
18     vector<int> head;
19     vector<int> subtree_size;
20     vector<int> level;
21     vector<int> heavy_child;
22     vector<ftype> subtree_weight;
23     vector<ftype> path_weight;
24     vector<vector<int>> adj;
25     vector<int> at;
26     Segtree seg = Segtree(0);
27     int cpos;
28     int n;
29     int root;
30
31     HLD() {}
32
33     HLD(int n, vector<vector<int>>& adj, int root = 1) {
34         : adj(adj), n(n), root(root) {
35             seg = Segtree(n);
36             cpos = 0;
37             at.assign(n, 0);
38             parent.assign(n, 0);
39             pos.assign(n, 0);
40             head.assign(n, 0);
41             subtree_size.assign(n, 1);
42             level.assign(n, 0);
43             heavy_child.assign(n, -1);
44             parent[root] = -1;
45             dfs(root, -1);
46             decompose(root, -1);
47         }
48     }
49

```

```

50 void dfs(int v, int p) {
51     parent[v] = p;
52     if (p != -1) level[v] = level[p] + 1;
53     for (auto u : adj[v]) {
54         if (u != p) {
55             dfs(u, v);
56             subtree_size[v] += subtree_size[u];
57             if (heavy_child[v] == -1 || subtree_size[u] >
58                 subtree_size[heavy_child[v]]) heavy_child[v] = u
59         }
60     }
61 }
62
63 void decompose(int v, int chead) {
64     // start a new path
65     if (chead == -1) chead = v;
66
67     // consecutive ids in the hld path
68     at[cpos] = v;
69     pos[v] = cpos++;
70     head[v] = chead;
71
72     // if not a leaf
73     if (heavy_child[v] != -1) decompose(heavy_child[v],
74         chead);
75
76     // light child
77     for (auto u : adj[v]){
78         // start new path
79         if (u != parent[v] && u != heavy_child[v])
80             decompose(u, -1);
81     }
82
83 ll query_path(int a, int b) {
84     if (a == b) return 0;
85     if(pos[a] < pos[b]) swap(a, b);
86
87     if(head[a] == head[b]) return seg.query(pos[b] +
88         1, pos[a]);
89     return seg.f(seg.query(pos[head[a]], pos[a]),
90         query_path(parent[head[a]], b));
91 }
92
93 ftype query_subtree(int a) {
94     if (subtree_size[a] == 1) return 0;
95     return seg.query(pos[a] + 1, pos[a] +
96         subtree_size[a] - 1);
97 }
98
99 void update_path(int a, int b, int x) {
100     if (a == b) return;
101     if(pos[a] < pos[b]) swap(a, b);
102
103     if(head[a] == head[b]) return (void)seg.update(
104         pos[b] + 1, pos[a], x);
105     seg.update(pos[head[a]], pos[a], x); update_path(
106         parent[head[a]], b, x);
107 }
108
109 void update_subtree(int a, int val) {
110     if (subtree_size[a] == 1) return;
111     seg.update(pos[a] + 1, pos[a] + subtree_size[a] -
112         1, val);
113
114 // vertex
115 void update(int a, int val) {
116     seg.update(pos[a], pos[a], val);
117 }
118
119 //edge

```



```

113 void update(int a, int b, int val) {
114     if (parent[a] == b) swap(a, b);
115     update(b, val);
116 }
117
118 int lca(int a, int b) {
119     if(pos[a] < pos[b]) swap(a, b);
120     return head[a] == head[b] ? b : lca(parent[head[a]], b);
121 }
122 };

```

## 8.26 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 }

```

## 9 Geometry

### 9.1 Shoelace Boundary

```

1 // Description
2 // Shoelace formula finds the area of a polygon
3 // Boundary points return the number of integer
4 // points on the edges of a polygon
5 // not counting the vertexes
6
7 // Problem
8 // https://codeforces.com/gym/101873/problem/G
9
10 // Complexity
11 // O(n)
12
13 // before dividing by two
14 int shoelace(vector<point> & points) {
15     int n = points.size();
16     vector<point> v(n + 2);
17
18     for (int i = 1; i <= n; i++) {
19         v[i] = points[i - 1];
20     }
21     v[n + 1] = points[0];

```

```

21 int sum = 0;
22 for (int i = 1; i <= n; i++) {
23     sum += (v[i].x * v[i + 1].y - v[i + 1].x * v[
24         i].y);
25 }
26
27 sum = abs(sum);
28 return sum;
29 }
30
31 int boundary_points(vector<point> & points) {
32     int n = points.size();
33     vector<point> v(n + 2);
34
35     for (int i = 1; i <= n; i++) {
36         v[i] = points[i - 1];
37     }
38     v[n + 1] = points[0];
39
40     int ans = 0;
41     for (int i = 1; i <= n; i++) {
42         if (v[i].x == v[i + 1].x) ans += abs(v[i].y -
43             v[i + 1].y) - 1;
44         else if (v[i].y == v[i + 1].y) ans += abs(v[i]
45             ].x - v[i + 1].x) - 1;
46         else ans += gcd(abs(v[i].x - v[i + 1].x), abs
47             (v[i].y - v[i + 1].y)) - 1;
48     }
49     return points.size() + ans;

```

### 9.2 Mindistpair

```

1 ll MinDistPair(vp &vet){
2     int n = vet.size();
3     sort(vet.begin(), vet.end());
4     set<point> s;
5
6     ll best_dist = LLINF;
7     int j=0;
8     for(int i=0;i<n;i++){
9         ll d = ceil(sqrt(best_dist));
10        while(j<n and vet[i].x-vet[j].x >= d){
11            s.erase(point(vet[j].y, vet[j].x));
12            j++;
13        }
14
15        auto it1 = s.lower_bound({vet[i].y - d, vet[i]
16            ].x});
17        auto it2 = s.upper_bound({vet[i].y + d, vet[i]
18            ].x});
19
20        for(auto it=it1; it!=it2; it++){
21            ll dx = vet[i].x - it->y;
22            ll dy = vet[i].y - it->x;
23            if(best_dist > dx*dx + dy*dy){
24                best_dist = dx*dx + dy*dy;
25                // vet[i] e inv(it)
26            }
27        }
28
29        s.insert(point(vet[i].y, vet[i].x));
30    }
31    return best_dist;
32 }

```

### 9.3 2d

```

1 #define vp vector<point>
2 #define ld long double
3 const ld EPS = 1e-6;

```

```

4 const ld PI = acos(-1);
5
6 // typedef ll cod;
7 // bool eq(cod a, cod b){ return (a==b); }
8 typedef ld cod;
9 bool eq(cod a, cod b){ return abs(a - b) <= EPS; }
10
11 struct point{
12     cod x, y;
13     int id;
14     point(cod x=0, cod y=0): x(x), y(y){}
15
16     point operator+(const point &o) const{ return {x+o.x, y+o.y}; }
17     point operator-(const point &o) const{ return {x-o.x, y-o.y}; }
18     point operator*(cod t) const{ return {x*t, y*t}; }
19     point operator/(cod t) const{ return {x/t, y/t}; }
20     cod operator*(const point &o) const{ return x * o.x + y * o.y; }
21     cod operator^(const point &o) const{ return x * o.y - y * o.x; }
22     bool operator<(const point &o) const{
23         return (eq(x, o.x) ? y < o.y : x < o.x);
24     }
25     bool operator==(const point &o) const{
26         return eq(x, o.x) and eq(y, o.y);
27     }
28     friend ostream& operator<<(ostream& os, point p) {
29         return os << "(" << p.x << ", " << p.y << ")"; }
30 };
31
32 int ccw(point a, point b, point e){ // -1=dir; 0=
33     collinear; 1=esq;
34     cod tmp = (b-a) ^ (e-a); // vector from a to b
35     return (tmp > EPS) - (tmp < -EPS);
36 }
37 ld norm(point a){ // Modulo
38     return sqrt(a * a);
39 }
40 cod norm2(point a){
41     return a * a;
42 }
43 bool nulo(point a){
44     return (eq(a.x, 0) and eq(a.y, 0));
45 }
46 point rotccw(point p, ld a){
47     // a = PI*a/180; // graus
48     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+p.x*sin(a)));
49 }
50 point rot90cw(point a) { return point(a.y, -a.x); }
51 point rot90ccw(point a) { return point(-a.y, a.x); }
52
53 ld proj(point a, point b){ // a sobre b
54     return a*b/norm(b);
55 }
56 ld angle(point a, point b){ // em radianos
57     ld ang = a*b / norm(a) / norm(b);
58     return acos(max(min(ang, (ld)1), (ld)-1));
59 }
60 ld angle_vec(point v){
61     // return 180/PI*atan2(v.x, v.y); // graus
62     return atan2(v.x, v.y);
63 }
64 ld order_angle(point a, point b){ // from a to b ccw
65     (a in front of b)
66     ld aux = angle(a,b)*180/PI;
67     return ((a^b)<=0 ? aux:360-aux);
68 }
69
70 bool angle_less(point a1, point b1, point a2, point
71     b2){ // ang(a1,b1) <= ang(a2,b2)
72     point p1((a1*b1), abs((a1^b1)));
73     point p2((a2*b2), abs((a2^b2)));
74     return (p1^p2) <= 0;
75 }
76
77 ld area(vp &p){ // (points sorted)
78     ld ret = 0;
79     for(int i=2;i<(int)p.size();i++)
80         ret += (p[i]-p[0])^(p[i-1]-p[0]);
81     return abs(ret/2);
82 }
83 ld areaT(point &a, point &b, point &c){
84     return abs((b-a)^(c-a))/2.0;
85 }
86
87 point center(vp &A){
88     point c = point();
89     int len = A.size();
90     for(int i=0;i<len;i++)
91         c=c+A[i];
92     return c/len;
93 }
94
95 point forca_mod(point p, ld m){
96     ld cm = norm(p);
97     if(cm<EPS) return point();
98     return point(p.x*m/cm,p.y*m/cm);
99 }
100
101 ld param(point a, point b, point v){
102     // v = t*(b-a) + a // return t;
103     // assert(line(a, b).inside_seg(v));
104     return ((v-a) * (b-a)) / ((b-a) * (b-a));
105 }
106
107 bool simetric(vp &a){ //ordered
108     int n = a.size();
109     point c = center(a);
110     if(n&1) return false;
111     for(int i=0;i<n/2;i++)
112         if(ccw(a[i], a[i+n/2], c) != 0)
113             return false;
114     return true;
115 }
116
117 point mirror(point m1, point m2, point p){
118     // mirror point p around segment m1m2
119     point seg = m2-m1;
120     ld t0 = ((p-m1)*seg) / (seg*seg);
121     point ort = m1 + seg*t0;
122     point pm = ort-(p-ort);
123     return pm;
124 }
125
126 // Line
127
128 struct line{
129     point p1, p2;
130     cod a, b, c; // ax+by+c = 0;
131     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
132     line(point p1=0, point p2=0): p1(p1), p2(p2){
133         a = p1.y - p2.y;
134         b = p2.x - p1.x;
135         c = p1 ^ p2;
136     }
137     line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
138     {
139         // Gera os pontos p1 p2 dados os coeficientes

```

```

139 // isso aqui eh um lixo mas quebra um galho
kkkkkk
140 if(b==0){
141     p1 = point(1, -c/a);
142     p2 = point(0, -c/a);
143 }else{
144     p1 = point(1, (-c-a*1)/b);
145     p2 = point(0, -c/b);
146 }
147 }
148
149 cod eval(point p){
150     return a*p.x+b*p.y+c;
151 }
152 bool inside(point p){
153     return eq(eval(p), 0);
154 }
155 point normal(){
156     return point(a, b);
157 }
158
159 bool inside_seg(point p){
160     return (
161         ((p1-p) ^ (p2-p)) == 0 and
162         ((p1-p) * (p2-p)) <= 0
163     );
164 }
165
166 };
167
168 // be careful with precision error
169 vp inter_line(line l1, line l2){
170     ld det = l1.a*l2.b - l1.b*l2.a;
171     if(det==0) return {};
172     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
173     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
174     return {point(x, y)};
175 }
176
177 // segments not collinear
178 vp inter_seg(line l1, line l2){
179     vp ans = inter_line(l1, l2);
180     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.
inside_seg(ans[0]))
181         return {};
182     return ans;
183 }
184 bool seg_has_inter(line l1, line l2){
185     // if collinear
186     if (l1.inside_seg(l2.p1) || l1.inside_seg(l2.p2)
|| l2.inside_seg(l1.p1) || l2.inside_seg(l1.p2))
187         return true;
188
189     return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.
p2, l2.p2) < 0 and
190         ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.
p2, l1.p2) < 0;
191 }
192 ld dist_seg(point p, point a, point b){ // point -
seg
193     if((p-a)*(b-a) < EPS) return norm(p-a);
194     if((p-b)*(a-b) < EPS) return norm(p-b);
195     return abs((p-a)^(b-a)) / norm(b-a);
196 }
197
198 ld dist_line(point p, line l){ // point - line
199     return abs(l.eval(p))/sqrt(1.a*1.a + 1.b*1.b);
200 }
201
202 line bisector(point a, point b){
203     point d = (b-a)*2;
204     return line(d.x, d.y, a*a - b*b);
205 }
206
207 line perpendicular(line l, point p){ // passes
through p
208     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
209 }
210
211
212 // Circle
213 // Circle
214
215 struct circle{
216     point c; cod r;
217     circle() : c(0, 0), r(0){}
218     circle(const point o) : c(o), r(0){}
219     circle(const point a, const point b){
220         c = (a+b)/2;
221         r = norm(a-c);
222     }
223
224     circle(const point a, const point b, const point
cc){
225         assert(ccw(a, b, cc) != 0);
226         c = inter_line(bisector(a, b), bisector(b, cc
))[0];
227         r = norm(a-c);
228     }
229     bool inside(const point &a) const{
230         return norm(a - c) <= r + EPS;
231     }
232 };
233
234 pair<point, point> tangent_points(circle cr, point p)
{
235     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
236     point p1 = rotccw(cr.c-p, -theta);
237     point p2 = rotccw(cr.c-p, theta);
238     assert(d1 >= cr.r);
239     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
240     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
241     return {p1, p2};
242 }
243
244 circle incircle(point p1, point p2, point p3){
245     ld m1 = norm(p2-p3);
246     ld m2 = norm(p1-p3);
247     ld m3 = norm(p1-p2);
248     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
249     ld s = 0.5*(m1+m2+m3);
250     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
251     return circle(c, r);
252 }
253
254 circle circumcircle(point a, point b, point c) {
255     circle ans;
256     point u = point((b-a).y, -(b-a).x);
257     point v = point((c-a).y, -(c-a).x);
258     point n = (c-b)*0.5;
259     ld t = (u^n)/(v^u);
260     ans.c = ((a+c)*0.5) + (v*t);
261     ans.r = norm(ans.c-a);
262     return ans;
263 }
264
265 vp inter_circle_line(circle C, line L){
266     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.
p1)*(ab) / (ab*ab));
267     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
/ (ab*ab);
268     if (h2 < -EPS) return {};
269     if (eq(h2, 0)) return {p};
270     point h = (ab/norm(ab)) * sqrt(h2);
271

```

```

272     return {p - h, p + h};
273 }
274
275 vp inter_circle(circle C1, circle C2){
276     if(C1.c == C2.c) { assert(C1.r != C2.r); return
    {};}
277     point vec = C2.c - C1.c;
278     ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
    ;
279     ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
    C1.r*C1.r - p*p*d2;
280     if (sum*sum < d2 or dif*dif > d2) return {};
281     point mid = C1.c + vec*p, per = point(-vec.y, vec
    .x) * sqrt(max((ld)0, h2) / d2);
282     if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
283     return {mid + per, mid - per};
284 }
285
286 // minimum circle cover O(n) amortizado
287 circle min_circle_cover(vp v){
288     random_shuffle(v.begin(), v.end());
289     circle ans;
290     int n = v.size();
291     for(int i=0;i<n;i++) if(!ans.inside(v[i])){
292         ans = circle(v[i]);
293         for(int j=0;j<i;j++) if(!ans.inside(v[j])){
294             ans = circle(v[i], v[j]);
295             for(int k=0;k<j;k++) if(!ans.inside(v[k]))
    ){
296                 ans = circle(v[i], v[j], v[k]);
297             }
298         }
299     }
300     return ans;
301 }

```

## 9.4 Inside Polygon

```

1 // Description
2 // Checks if a given point is inside, outside or on
  the boundary of a polygon
3
4 // Problem
5 // https://cses.fi/problemset/task/2192/
6
7 // Complexity
8 // O(n)
9
10 int inside(vp &p, point pp){
11     // 1 - inside / 0 - boundary / -1 - outside
12     int n = p.size();
13     for(int i=0;i<n;i++){
14         int j = (i+1)%n;
15         if(line({p[i], p[j]}).inside_seg(pp))
16             return 0; // boundary
17     }
18     int inter = 0;
19     for(int i=0;i<n;i++){
20         int j = (i+1)%n;
21         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
    [i], p[j], pp)==1)
22             inter++; // up
23         else if(p[j].x <= pp.x and pp.x < p[i].x and
    ccw(p[i], p[j], pp)==-1)
24             inter++; // down
25     }
26
27     if(inter%2==0) return -1; // outside
28     else return 1; // inside
29 }

```

## 9.5 Convexhull

```

1 // Graham Scan
2 struct pt {
3     double x, y;
4     bool operator == (pt const& t) const {
5         return x == t.x && y == t.y;
6     }
7 };
8
9 int orientation(pt a, pt b, pt c) {
10     double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b
    .y);
11     if (v < 0) return -1; // clockwise
12     if (v > 0) return +1; // counter-clockwise
13     return 0;
14 }
15
16 bool cw(pt a, pt b, pt c, bool include_collinear) {
17     int o = orientation(a, b, c);
18     return o < 0 || (include_collinear && o == 0);
19 }
20 bool collinear(pt a, pt b, pt c) { return orientation
    (a, b, c) == 0; }
21
22 void convex_hull(vector<pt>& a, bool
    include_collinear = false) {
23     pt p0 = *min_element(a.begin(), a.end(), [](pt a,
    pt b) {
24         return make_pair(a.y, a.x) < make_pair(b.y, b
    .x);
25     });
26     sort(a.begin(), a.end(), [&p0](const pt& a, const
    pt& b) {
27         int o = orientation(p0, a, b);
28         if (o == 0)
29             return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)
    *(p0.y-a.y)
30                 < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)
    *(p0.y-b.y);
31         return o < 0;
32     });
33     if (include_collinear) {
34         int i = (int)a.size()-1;
35         while (i >= 0 && collinear(p0, a[i], a.back()
    )) i--;
36         reverse(a.begin()+i+1, a.end());
37     }
38
39     vector<pt> st;
40     for (int i = 0; i < (int)a.size(); i++) {
41         while (st.size() > 1 && !cw(st[st.size()-2],
    st.back(), a[i], include_collinear))
42             st.pop_back();
43         st.push_back(a[i]);
44     }
45
46     if (include_collinear == false && st.size() == 2
    && st[0] == st[1])
47         st.pop_back();
48
49     a = st;
50 }
51
52 // Monotone Chain
53 struct pt {
54     double x, y;
55 };
56
57 int orientation(pt a, pt b, pt c) {
58     double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b
    .y);
59     if (v < 0) return -1; // clockwise
60     if (v > 0) return +1; // counter-clockwise
61 }

```

```

62     return 0;
63 }
64
65 bool cw(pt a, pt b, pt c, bool include_collinear) {
66     int o = orientation(a, b, c);
67     return o < 0 || (include_collinear && o == 0);
68 }
69 bool ccw(pt a, pt b, pt c, bool include_collinear) {
70     int o = orientation(a, b, c);
71     return o > 0 || (include_collinear && o == 0);
72 }
73
74 void convex_hull(vector<pt>& a, bool
75     include_collinear = false) {
76     if (a.size() == 1)
77         return;
78
79     sort(a.begin(), a.end(), [](pt a, pt b) {
80         return make_pair(a.x, a.y) < make_pair(b.x, b
81             .y);
82     });
83     pt p1 = a[0], p2 = a.back();
84     vector<pt> up, down;
85     up.push_back(p1);
86     down.push_back(p1);
87     for (int i = 1; i < (int)a.size(); i++) {
88         if (i == a.size() - 1 || cw(p1, a[i], p2,
89             include_collinear)) {
90             while (up.size() >= 2 && !cw(up[up.size()
91                 -2], up[up.size()-1], a[i], include_collinear))
92                 up.pop_back();
93             up.push_back(a[i]);
94         }
95         if (i == a.size() - 1 || ccw(p1, a[i], p2,
96             include_collinear)) {
97             while (down.size() >= 2 && !ccw(down[down
98                 .size()-2], down[down.size()-1], a[i],
99                 include_collinear))
100                 down.pop_back();
101             down.push_back(a[i]);
102         }
103     }
104     if (include_collinear && up.size() == a.size()) {
105         reverse(a.begin(), a.end());
106         return;
107     }
108     a.clear();
109     for (int i = 0; i < (int)up.size(); i++)
110         a.push_back(up[i]);
111     for (int i = down.size() - 2; i > 0; i--)
112         a.push_back(down[i]);
113 }

```

## 9.6 Delaunay

```

1  typedef long long ll;
2
3  bool ge(const ll& a, const ll& b) { return a >= b; }
4  bool le(const ll& a, const ll& b) { return a <= b; }
5  bool eq(const ll& a, const ll& b) { return a == b; }
6  bool gt(const ll& a, const ll& b) { return a > b; }
7  bool lt(const ll& a, const ll& b) { return a < b; }
8  int sgn(const ll& a) { return a >= 0 ? a > 1 : 0 :
9      -1; }
10
11 struct pt {
12     ll x, y;
13     pt() {}
14     pt(ll _x, ll _y) : x(_x), y(_y) {}
15     pt operator-(const pt& p) const {
16         return pt(x - p.x, y - p.y);
17     }
18 }

```

```

17 ll cross(const pt& p) const {
18     return x * p.y - y * p.x;
19 }
20 ll cross(const pt& a, const pt& b) const {
21     return (a - *this).cross(b - *this);
22 }
23 ll dot(const pt& p) const {
24     return x * p.x + y * p.y;
25 }
26 ll dot(const pt& a, const pt& b) const {
27     return (a - *this).dot(b - *this);
28 }
29 ll sqrLength() const {
30     return this->dot(*this);
31 }
32 bool operator==(const pt& p) const {
33     return eq(x, p.x) && eq(y, p.y);
34 }
35 };
36
37 const pt inf_pt = pt(1e18, 1e18);
38
39 struct QuadEdge {
40     pt origin;
41     QuadEdge* rot = nullptr;
42     QuadEdge* onext = nullptr;
43     bool used = false;
44     QuadEdge* rev() const {
45         return rot->rot;
46     }
47     QuadEdge* lnext() const {
48         return rot->rev()->onext->rot;
49     }
50     QuadEdge* oprev() const {
51         return rot->onext->rot;
52     }
53     pt dest() const {
54         return rev()->origin;
55     }
56 };
57
58 QuadEdge* make_edge(pt from, pt to) {
59     QuadEdge* e1 = new QuadEdge;
60     QuadEdge* e2 = new QuadEdge;
61     QuadEdge* e3 = new QuadEdge;
62     QuadEdge* e4 = new QuadEdge;
63     e1->origin = from;
64     e2->origin = to;
65     e3->origin = e4->origin = inf_pt;
66     e1->rot = e3;
67     e2->rot = e4;
68     e3->rot = e2;
69     e4->rot = e1;
70     e1->onext = e1;
71     e2->onext = e2;
72     e3->onext = e4;
73     e4->onext = e3;
74     return e1;
75 }
76
77 void splice(QuadEdge* a, QuadEdge* b) {
78     swap(a->onext->rot->onext, b->onext->rot->onext);
79     swap(a->onext, b->onext);
80 }
81
82 void delete_edge(QuadEdge* e) {
83     splice(e, e->oprev());
84     splice(e->rev(), e->rev()->oprev());
85     delete e->rev()->rot;
86     delete e->rev();
87     delete e->rot;
88     delete e;
89 }

```

```

90
91 QuadEdge* connect(QuadEdge* a, QuadEdge* b) {
92     QuadEdge* e = make_edge(a->dest(), b->origin());
93     splice(e, a->lnext());
94     splice(e->rev(), b);
95     return e;
96 }
97
98 bool left_of(pt p, QuadEdge* e) {
99     return gt(p.cross(e->origin, e->dest()), 0);
100 }
101
102 bool right_of(pt p, QuadEdge* e) {
103     return lt(p.cross(e->origin, e->dest()), 0);
104 }
105
106 template <class T>
107 T det3(T a1, T a2, T a3, T b1, T b2, T b3, T c1, T c2,
108        T c3) {
109     return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 -
110        c1 * b3) +
111        a3 * (b1 * c2 - c1 * b2);
112 }
113
114 bool in_circle(pt a, pt b, pt c, pt d) {
115     // If there is __int128, calculate directly.
116     // Otherwise, calculate angles.
117     #if defined(__LP64__) || defined(_WIN64)
118         __int128 det = -det3<__int128>(b.x, b.y, b.
119             sqrLength(), c.x, c.y,
120             c.sqrLength(), d.x,
121             d.y, d.sqrLength());
122         det += det3<__int128>(a.x, a.y, a.sqrLength(), c.
123             x, c.y, c.sqrLength(), d.x,
124             d.y, d.sqrLength());
125         det -= det3<__int128>(a.x, a.y, a.sqrLength(), b.
126             x, b.y, b.sqrLength(), d.x,
127             d.y, d.sqrLength());
128         det += det3<__int128>(a.x, a.y, a.sqrLength(), b.
129             x, b.y, b.sqrLength(), c.x,
130             c.y, c.sqrLength());
131         return det > 0;
132     #else
133         auto ang = [(pt l, pt mid, pt r) {
134             ll x = mid.dot(l, r);
135             ll y = mid.cross(l, r);
136             long double res = atan2((long double)x, (long
137                 double)y);
138             return res;
139         }];
140         long double kek = ang(a, b, c) + ang(c, d, a) -
141             ang(b, c, d) - ang(d, a, b);
142         if (kek > 1e-8)
143             return true;
144         else
145             return false;
146     #endif
147 }
148
149 pair<QuadEdge*, QuadEdge*> build_tr(int l, int r,
150     vector<pt>& p) {
151     if (r - l + 1 == 2) {
152         QuadEdge* res = make_edge(p[l], p[r]);
153         return make_pair(res, res->rev());
154     }
155     if (r - l + 1 == 3) {
156         QuadEdge* a = make_edge(p[l], p[l + 1]), *b =
157             make_edge(p[l + 1], p[r]);
158         splice(a->rev(), b);
159         int sg = sgn(p[l].cross(p[l + 1], p[r]));
160         if (sg == 0)
161             return make_pair(a, b->rev());
162         QuadEdge* c = connect(b, a);
163     }
164     if (sg == 1)
165         return make_pair(a, b->rev());
166     else
167         return make_pair(c->rev(), c);
168 }
169
170 int mid = (l + r) / 2;
171 QuadEdge* ldo, *ldi, *rdo, *rdi;
172 tie(ldo, ldi) = build_tr(l, mid, p);
173 tie(rdi, rdo) = build_tr(mid + 1, r, p);
174 while (true) {
175     if (left_of(rdi->origin, ldi)) {
176         ldi = ldi->lnext();
177         continue;
178     }
179     if (right_of(ldi->origin, rdi)) {
180         rdi = rdi->rev()->onext;
181         continue;
182     }
183     break;
184 }
185
186 QuadEdge* basel = connect(rdi->rev(), ldi);
187 auto valid = [&basel](QuadEdge* e) { return
188     right_of(e->dest(), basel); };
189 if (ldi->origin == ldo->origin)
190     ldo = basel->rev();
191 if (rdi->origin == rdo->origin)
192     rdo = basel;
193 while (true) {
194     QuadEdge* lcand = basel->rev()->onext;
195     if (valid(lcand)) {
196         while (in_circle(basel->dest(), basel->
197             origin, lcand->dest(),
198                 lcand->onext->dest())) {
199             QuadEdge* t = lcand->onext;
200             delete_edge(lcand);
201             lcand = t;
202         }
203     }
204     QuadEdge* rcand = basel->oprev();
205     if (valid(rcand)) {
206         while (in_circle(basel->dest(), basel->
207             origin, rcand->dest(),
208                 rcand->oprev()->dest())) {
209             QuadEdge* t = rcand->oprev();
210             delete_edge(rcand);
211             rcand = t;
212         }
213     }
214     if (!valid(lcand) && !valid(rcand))
215         break;
216     if (!valid(lcand) ||
217         (valid(rcand) && in_circle(lcand->dest(),
218             lcand->origin,
219                 rcand->origin,
220                     rcand->dest())))
221         basel = connect(rcand, basel->rev());
222     else
223         basel = connect(basel->rev(), lcand->rev
224             ());
225 }
226 return make_pair(ldo, rdo);
227 }
228
229 vector<tuple<pt, pt, pt>> delaunay(vector<pt> p) {
230     sort(p.begin(), p.end(), [](const pt& a, const pt
231         & b) {
232         return lt(a.x, b.x) || (eq(a.x, b.x) && lt(a.
233             y, b.y));
234     });
235     auto res = build_tr(0, (int)p.size() - 1, p);
236     QuadEdge* e = res.first;
237     vector<QuadEdge*> edges = {e};

```

```

216 while (lt(e->onext->dest().cross(e->dest(), e->
origin), 0))
217     e = e->onext;
218 auto add = [&p, &e, &edges]() {
219     QuadEdge* curr = e;
220     do {
221         curr->used = true;
222         p.push_back(curr->origin);
223         edges.push_back(curr->rev());
224         curr = curr->lnext();
225     } while (curr != e);
226 };
227 add();
228 p.clear();
229 int kek = 0;
230 while (kek < (int)edges.size()) {
231     if (!(e = edges[kek++])->used)
232         add();
233 }
234 vector<tuple<pt, pt, pt>> ans;
235 for (int i = 0; i < (int)p.size(); i += 3) {
236     ans.push_back(make_tuple(p[i], p[i + 1], p[i
+ 2]));
237 }
238 return ans;
239 }

```

## 9.7 Closest Pair Points

```

1 // Description
2 // Find the squared distance between the closest two
   points among n points
3 // Also finds which pair of points is closest (could
   be more than one)
4
5 // Problem
6 // https://cses.fi/problemset/task/2194/
7
8 // Complexity
9 // O(n log n)

```

```

10
11 ll closest_pair_points(vp &vet){
12     pair<point, point> ans;
13     int n = vet.size();
14     sort(vet.begin(), vet.end());
15     set<point> s;
16
17     ll best_dist = LLONG_MAX;
18     int j=0;
19     for(int i=0;i<n;i++){
20         ll d = ceil(sqrt(best_dist));
21         while(j<n and vet[i].x-vet[j].x >= d){
22             s.erase(point(vet[j].y, vet[j].x));
23             j++;
24         }
25
26         auto it1 = s.lower_bound({vet[i].y - d, vet[i
].x});
27         auto it2 = s.upper_bound({vet[i].y + d, vet[i
].x});
28
29         for(auto it=it1; it!=it2; it++){
30             ll dx = vet[i].x - it->y;
31             ll dy = vet[i].y - it->x;
32
33             if(best_dist > dx*dx + dy*dy){
34                 best_dist = dx*dx + dy*dy;
35                 // closest pair points
36                 ans = mp(vet[i], point(it->y, it->x))
37             }
38         }
39
40         s.insert(point(vet[i].y, vet[i].x));
41     }
42
43     // best distance squared
44     return best_dist;
45 }

```