

# Competitive Programming Notebook

As Meninas Superpoderosas

## Contents

<b>1 DS</b>	<b>2</b>	<b>7 General</b>	<b>15</b>
1.1 Ordered Set . . . . .	2	7.1 Mix Hash . . . . .	15
1.2 Bigk . . . . .	2	7.2 Xor 1 To N . . . . .	15
1.3 Mex . . . . .	2	7.3 Base Converter . . . . .	15
1.4 Segtree Lazy Iterative . . . . .	3	7.4 Min Priority Queue . . . . .	16
1.5 Kruskal . . . . .	3	7.5 Input By File . . . . .	16
1.6 Dsu . . . . .	4	7.6 Template . . . . .	16
<b>2 Graph</b>	<b>4</b>	7.7 First True . . . . .	16
2.1 Dijkstra . . . . .	4	7.8 Get Subsets Sum Iterative . . . . .	16
2.2 Ford Fulkerson . . . . .	4	7.9 Last True . . . . .	16
2.3 2sat . . . . .	5	7.10 Interactive . . . . .	17
2.4 Floyd Warshall . . . . .	6	7.11 Next Permutation . . . . .	17
2.5 Lca . . . . .	6	7.12 Random . . . . .	17
2.6 Bfs . . . . .	6	<b>8 Primitives</b>	<b>17</b>
2.7 Min Cost Max Flow . . . . .	7		
2.8 Has Negative Cycle . . . . .	8		
2.9 Dinic . . . . .	8		
<b>3 String</b>	<b>9</b>		
3.1 Trie Xor . . . . .	9		
3.2 Split . . . . .	10		
3.3 Is Substring . . . . .	10		
3.4 Hash . . . . .	10		
<b>4 Math</b>	<b>11</b>		
4.1 Fft Quirino . . . . .	11		
4.2 Ceil . . . . .	11		
4.3 Division Trick . . . . .	11		
4.4 Fexp . . . . .	11		
4.5 Sieve . . . . .	12		
4.6 Divisors . . . . .	12		
4.7 Log Any Base . . . . .	12		
4.8 Generate Primes . . . . .	12		
4.9 Factorization . . . . .	12		
4.10 Is Prime . . . . .	12		
<b>5 Geometry</b>	<b>12</b>		
5.1 Convex Hull . . . . .	12		
<b>6 DP</b>	<b>13</b>		
6.1 Edit Distance . . . . .	13		
6.2 Range Dp . . . . .	13		
6.3 Lis Segtree . . . . .	14		
6.4 Digit Dp 2 . . . . .	14		
6.5 Lis Binary Search . . . . .	14		
6.6 Lcs . . . . .	14		
6.7 Digit Dp . . . . .	15		

# 1 DS

## 1.1 Ordered Set

```

1 // Ordered Set
2 //
3 // set roubado com mais operacoes
4 //
5 // para alterar para multiset
6 // trocar less para less_equal
7 //
8 // ordered_set<int> s
9 //
10 // order_of_key(k) // number of items strictly
11 // smaller than k -> int
12 //
13 // find_by_order(k) // k-th element in a set (
14 // counting from zero) -> iterator
15 //
16 // https://cses.fi/problemset/task/2169
17 //
18 // O(log N) para insert, erase (com iterator),
19 // order_of_key, find_by_order
20
21 using namespace __gnu_pbds;
22 template <typename T>
23 using ordered_set = tree<T,null_type,less<T>,
24 rb_tree_tag,tree_order_statistics_node_update>;
25
26 void erase(ordered_set& a, int x){
27     int r = a.order_of_key(x);
28     auto it = a.find_by_order(r);
29     a.erase(it);
30 }

```

## 1.2 Bigk

```

1 struct SetSum {
2     ll sum;
3     multiset<ll> ms;
4
5     SetSum() {}
6
7     void add(ll x) {
8         sum += x;
9         ms.insert(x);
10    }
11
12    int rem(ll x) {
13        auto it = ms.find(x);
14
15        if (it == ms.end()) {
16            return 0;
17        }
18
19        sum -= x;
20        ms.erase(it);
21        return 1;
22    }
23
24    ll getMin() { return *ms.begin(); }
25
26    ll getMax() { return *ms.rbegin(); }
27
28    ll getSum() { return sum; }
29
30    int size() { return (int)ms.size(); }
31 };
32
33 struct BigK {
34     int k;
35     SetSum gt, mt;
36

```

```

37     BigK(int k): k(k) {}
38
39     void balance() {
40         while (gt.size() > k) {
41             ll mn = gt.getMin();
42             gt.rem(mn);
43             mt.add(mn);
44         }
45
46         while (gt.size() < k && mt.size() > 0) {
47             ll mx = mt.getMax();
48             mt.rem(mx);
49             gt.add(mx);
50         }
51     }
52
53     void add(ll x) {
54         gt.add(x);
55         balance();
56     }
57
58     void rem(ll x) {
59         if (mt.rem(x) == 0) {
60             gt.rem(x);
61         }
62
63         balance();
64     }
65
66     // be careful, O(abs(oldK - newk) * log)
67     void setK(int _k) {
68         k = _k;
69         balance();
70     }
71
72     // O(log)
73     void incK() { setK(k + 1); }
74
75     // O(log)
76     void decK() { setK(k - 1); }
77 };

```

## 1.3 Mex

```

1 // Mex
2 //
3 // facilita queries de mex com update
4 //
5 // N eh o maior valor possivel do mex
6 // add(x) = adiciona x
7 // rem(x) = remove x
8 //
9 // O(log N) por insert
10 // O(1) por query
11
12 struct Mex {
13     map<int, int> cnt;
14     set<int> possible;
15
16     Mex(int n) {
17         for (int i = 0; i <= n + 1; i++) {
18             possible.insert(i);
19         }
20     }
21
22     void add(int x) {
23         cnt[x]++;
24         possible.erase(x);
25     }
26
27     void rem(int x) {
28         cnt[x]--;
29

```

```

30         if (cnt[x] == 0) {
31             possible.insert(x);
32         }
33     }
34
35     int query() {
36         return *(possible.begin());
37     }
38 };

```

## 1.4 Segtree Lazy Iterative

```

1 // Segtree iterativa com lazy
2 //
3 // https://codeforces.com/gym/103708/problem/C
4 //
5 // O(N * log(N)) build
6 // O(log(N)) update e query
7
8 const int MAX = 524288; // NEED TO BE POWER OF 2 !!!
9 const int LOG = 19; // LOG = ceil(log2(MAX))
10
11 namespace seg {
12     ll seg[2*MAX], lazy[2*MAX];
13     int n;
14
15     ll junta(ll a, ll b) {
16         return a+b;
17     }
18
19     // soma x na posicao p de tamanho tam
20     void poe(int p, ll x, int tam, bool prop=1) {
21         seg[p] += x*tam;
22         if (prop and p < n) lazy[p] += x;
23     }
24
25     // atualiza todos os pais da folha p
26     void sobe(int p) {
27         for (int tam = 2; p /= 2; tam *= 2) {
28             seg[p] = junta(seg[2*p], seg[2*p+1]);
29             poe(p, lazy[p], tam, 0);
30         }
31     }
32
33     void upd_lazy(int i, int tam) {
34         if (lazy[i] && (2 * i + 1) < 2 * MAX) {
35             poe(2*i, lazy[i], tam);
36             poe(2*i+1, lazy[i], tam);
37             lazy[i] = 0;
38         }
39     }
40
41     // propaga o caminho da raiz ate a folha p
42     void prop(int p) {
43         int tam = 1 << (LOG-1);
44         for (int s = LOG; s; s--, tam /= 2) {
45             int i = p >> s;
46             upd_lazy(i, tam);
47         }
48     }
49
50     void build(int n2) {
51         n = n2;
52         for (int i = 0; i < n; i++) seg[n+i] = 0;
53         for (int i = n-1; i; i--) seg[i] = junta(seg[2*i], seg[2*i+1]);
54         for (int i = 0; i < 2*n; i++) lazy[i] = 0;
55     }
56
57     ll query(int a, int b) {
58         ll ret = 0;
59         for (prop(a+=n), prop(b+=n); a <= b; ++a/=2, --b/=2) {

```

```

60             if (a%2 == 1) ret = junta(ret, seg[a]);
61             if (b%2 == 0) ret = junta(ret, seg[b]);
62         }
63         return ret;
64     }
65
66     void update(int a, int b, int x) {
67         int a2 = a += n, b2 = b += n, tam = 1;
68         for (; a <= b; ++a/=2, --b/=2, tam *= 2) {
69             if (a%2 == 1) poe(a, x, tam);
70             if (b%2 == 0) poe(b, x, tam);
71         }
72         sobe(a2), sobe(b2);
73     }
74
75     int findkth(int x, int l, int r, ll k, int tam){
76         int esq = x + x;
77         int dir = x + x + 1;
78
79         upd_lazy(x, tam);
80         upd_lazy(esq, tam/2);
81         upd_lazy(dir, tam/2);
82
83         if(l == r){
84             return l;
85         } else {
86             int mid = l + (r-l)/2;
87
88             if(seg[esq] >= k){
89                 return findkth(esq,l,mid,k, tam/2);
90             } else {
91                 return findkth(dir,mid+1, r, k - seg[
92                     esq], tam/2);
93             }
94         }
95
96         int findkth(ll k){
97             // kth smallest, O(logN)
98             // use position i to count how many times
99             // value 'i' appear
100             // merge must be the sum of nodes
101             return findkth(1,0,n-1,k,(1 << (LOG-1)));
102         }
103     };

```

## 1.5 Kruskal

```

1 struct Edge {
2     int u, v;
3     ll weight;
4
5     Edge() {}
6
7     Edge(int u, int v, ll weight) : u(u), v(v),
8         weight(weight) {}
9
10    bool operator<(Edge const& other) {
11        return weight < other.weight;
12    }
13 };
14
15 vector<Edge> kruskal(vector<Edge> edges, int n) {
16     vector<Edge> result;
17     ll cost = 0;
18
19     sort(edges.begin(), edges.end());
20     DSU dsu(n);
21
22     for (auto e : edges) {
23         if (!dsu.same(e.u, e.v)) {
24             cost += e.weight;
25             result.push_back(e);
26         }
27     }
28 }

```

```

25         dsu.unite(e.u, e.v);
26     }
27 }
28
29 return result;
30 }

```

## 1.6 Dsu

```

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 };

```

## 2 Graph

### 2.1 Dijkstra

```

1 const int INF = 1e9+17;
2 vector<vector<pair<int, int>>> adj; // {neighbor,
3     weight}
4
5 void dijkstra(int s, vector<int> & d, vector<int> & p
6 ) {
7     int n = adj.size();
8     d.assign(n, INF);
9     p.assign(n, -1);
10
11    d[s] = 0;
12    set<pair<int, int>> q;
13    q.insert({0, s});
14    while (!q.empty()) {
15        int v = q.begin()->second;
16        q.erase(q.begin());
17
18        for (auto edge : adj[v]) {
19            int to = edge.first;
20            int len = edge.second;

```

```

19
20        if (d[v] + len < d[to]) {
21            q.erase({d[to], to});
22            d[to] = d[v] + len;
23            p[to] = v;
24            q.insert({d[to], to});
25        }
26    }
27 }
28 }

```

### 2.2 Ford Fulkerson

```

1 // Ford-Fulkerson
2 //
3 // max-flow / min-cut
4 //
5 // MAX nÃss
6 //
7 // https://cses.fi/problemset/task/1694/
8 //
9 // O(m * max_flow)
10
11 using ll = long long;
12 const int MAX = 510;
13
14 struct Flow {
15     int n;
16     ll adj[MAX][MAX];
17     bool used[MAX];
18
19     Flow(int n) : n(n) {};
20
21     void add_edge(int u, int v, ll c) {
22         adj[u][v] += c;
23         adj[v][u] = 0; // cuidado com isso
24     }
25
26     ll dfs(int x, int t, ll amount) {
27         used[x] = true;
28
29         if (x == t) return amount;
30
31         for (int i = 1; i <= n; i++) {
32             if (adj[x][i] > 0 && !used[i]) {
33                 ll sent = dfs(i, t, min(amount, adj[x
34 ][i]));
35
36                 if (sent > 0) {
37                     adj[x][i] -= sent;
38                     adj[i][x] += sent;
39
40                     return sent;
41                 }
42             }
43         }
44
45         return 0;
46     }
47
48     ll max_flow(int s, int t) { // source and sink
49         ll total = 0;
50         ll sent = -1;
51
52         while (sent != 0) {
53             memset(used, 0, sizeof(used));
54             sent = dfs(s, t, INT_MAX);
55             total += sent;
56         }
57
58         return total;
59 };

```

## 2.3 2sat

```

1 // 2SAT
2 //
3 // verifica se existe e encontra solu  o
4 // para f rmulas booleanas da forma
5 // (a or b) and (!a or c) and (...)
6 //
7 // indexado em 0
8 // n(a) = 2*x e n(~a) = 2*x+1
9 // a = 2 ; n(a) = 4 ; n(~a) = 5 ; n(a)^1 = 5 ; n(~a)
10 // ^1 = 4
11 // https://cses.fi/problemset/task/1684/
12 // https://codeforces.com/gym/104120/problem/E
13 // (add_eq, add_true, add_false e at_most_one n o
14 // foram testadas)
15 // O(n + m)
16
17 struct sat {
18     int n, tot;
19     vector<vector<int>> adj, adjt; // grafo original,
20     vector<int> vis, comp, ans;
21     stack<int> topo; // ordem topol gica
22
23     sat() {}
24     sat(int n_) : n(n_), tot(n), adj(2*n), adjt(2*n) {}
25
26     void dfs(int x) {
27         vis[x] = true;
28
29         for (auto e : adj[x]) {
30             if (!vis[e]) dfs(e);
31         }
32
33         topo.push(x);
34     }
35
36     void dfst(int x, int& id) {
37         vis[x] = true;
38         comp[x] = id;
39
40         for (auto e : adjt[x]) {
41             if (!vis[e]) dfst(e, id);
42         }
43     }
44
45     void add_impl(int a, int b) { // a -> b = (!a or
46         b)
47         a = (a >= 0 ? 2*a : -2*a-1);
48         b = (b >= 0 ? 2*b : -2*b-1);
49
50         adj[a].push_back(b);
51         adj[b^1].push_back(a^1);
52
53         adjt[b].push_back(a);
54         adjt[a^1].push_back(b^1);
55     }
56
57     void add_or(int a, int b) { // a or b
58         add_impl(~a, b);
59     }
60
61     void add_nor(int a, int b) { // a nor b = !(a or
62         b)
63         add_or(~a, b), add_or(a, ~b), add_or(~a, ~b);
64     }
65
66     void add_and(int a, int b) { // a and b
67         add_or(a, b), add_or(~a, b), add_or(a, ~b);
68     }
69
70     void add_nand(int a, int b) { // a nand b = !(a
71         and b)
72         add_or(~a, ~b);
73     }
74
75     void add_xor(int a, int b) { // a xor b = (a != b)
76         add_or(a, b), add_or(~a, ~b);
77     }
78
79     void add_xnor(int a, int b) { // a xnor b = !(a
80         xor b) = (a == b)
81         add_xor(~a, b);
82     }
83
84     void add_true(int a) { // a = T
85         add_or(a, ~a);
86     }
87
88     void add_false(int a) { // a = F
89         add_and(a, ~a);
90     }
91
92     // magia - brunomaletta
93     void add_true_old(int a) { // a = T (n sei se
94         funciona)
95         add_impl(~a, a);
96     }
97
98     void at_most_one(vector<int> v) { // no max um
99         verdadeiro
100         adj.resize(2*(tot+v.size()));
101         for (int i = 0; i < v.size(); i++) {
102             add_impl(tot+i, ~v[i]);
103             if (i) {
104                 add_impl(tot+i, tot+i-1);
105                 add_impl(v[i], tot+i-1);
106             }
107             tot += v.size();
108         }
109     }
110
111     pair<bool, vector<int>> solve() {
112         ans.assign(n, -1);
113         comp.assign(2*tot, -1);
114         vis.assign(2*tot, 0);
115         int id = 1;
116
117         for (int i = 0; i < 2*tot; i++) if (!vis[i])
118             dfs(i);
119
120         vis.assign(2*tot, 0);
121         while (topo.size()) {
122             auto x = topo.top();
123             topo.pop();
124
125             if (!vis[x]) {
126                 dfst(x, id);
127                 id++;
128             }
129         }
130
131         for (int i = 0; i < tot; i++) {
132             if (comp[2*i] == comp[2*i+1]) return {
133                 false, {} };
134             ans[i] = (comp[2*i] > comp[2*i+1]);
135         }
136
137         return {true, ans};
138     }
139 }

```

## 2.4 Floyd Warshall

```

1 const long long LLINF = 0x3f3f3f3f3f3f3fLL;
2
3 for (int i = 0; i < n; i++) {
4     for (int j = 0; j < n; j++) {
5         adj[i][j] = 0;
6     }
7 }
8
9 long long dist[MAX][MAX];
10 for (int i = 0; i < n; i++) {
11     for (int j = 0; j < n; j++) {
12         if (i == j)
13             dist[i][j] = 0;
14         else if (adj[i][j])
15             dist[i][j] = adj[i][j];
16         else
17             dist[i][j] = LLINF;
18     }
19 }
20
21 for (int k = 0; k < n; k++) {
22     for (int i = 0; i < n; i++) {
23         for (int j = 0; j < n; j++) {
24             dist[i][j] = min(dist[i][j], dist[i][k] +
25                             dist[k][j]);
26         }
27 }

```

## 2.5 Lca

```

1 // LCA
2 //
3 // lowest common ancestor between two nodes
4 //
5 // edit_distance(n, adj, root)
6 //
7 // https://cses.fi/problemset/task/1688
8 //
9 // O(log N)
10
11 struct LCA {
12     const int MAXE = 31;
13     vector<vector<int>> up;
14     vector<int> dep;
15
16     LCA(int n, vector<vector<int>>& adj, int root = 1) {
17         up.assign(n+1, vector<int>(MAXE, -1));
18         dep.assign(n+1, 0);
19
20         dep[root] = 1;
21         dfs(root, -1, adj);
22
23         for (int j = 1; j < MAXE; j++) {
24             for (int i = 1; i <= n; i++) {
25                 if (up[i][j-1] != -1)
26                     up[i][j] = up[ up[i][j-1] ][j-1];
27             }
28         }
29     }
30
31     void dfs(int x, int p, vector<vector<int>>& adj) {
32         up[x][0] = p;
33         for (auto e : adj[x]) {
34             if (e != p) {
35                 dep[e] = dep[x] + 1;
36                 dfs(e, x, adj);
37             }
38         }
39     }

```

```

38     }
39 }
40
41 int jump(int x, int k) { // jump from node x k
42     times
43     for (int i = 0; i < MAXE; i++) {
44         if (k & (1 << i) && x != -1) x = up[x][i];
45     }
46     return x;
47 }
48
49 int lca(int a, int b) {
50     if (dep[a] > dep[b]) swap(a, b);
51     b = jump(b, dep[b] - dep[a]);
52
53     if (a == b) return a;
54
55     for (int i = MAXE-1; i >= 0; i--) {
56         if (up[a][i] != up[b][i]) {
57             a = up[a][i];
58             b = up[b][i];
59         }
60     }
61
62     return up[a][0];
63 }
64
65 int dist(int a, int b) {
66     return dep[a] + dep[b] - 2 * dep[lca(a, b)];
67 }

```

## 2.6 Bfs

```

1 vector<vector<int>> adj; // adjacency list
2 // representation
3 int n; // number of nodes
4 int s; // source vertex
5
6 queue<int> q;
7 vector<bool> used(n + 1);
8 vector<int> d(n + 1), p(n + 1);
9
10 q.push(s);
11 used[s] = true;
12 p[s] = -1;
13 while (!q.empty()) {
14     int v = q.front();
15     q.pop();
16     for (int u : adj[v]) {
17         if (!used[u]) {
18             used[u] = true;
19             q.push(u);
20             d[u] = d[v] + 1;
21             p[u] = v;
22         }
23     }
24 }
25
26 // restore path
27 if (!used[u]) {
28     cout << "No path!";
29 } else {
30     vector<int> path;
31
32     for (int v = u; v != -1; v = p[v])
33         path.push_back(v);
34
35     reverse(path.begin(), path.end());
36
37     cout << "Path: ";
38     for (int v : path)
39         cout << v << " ";

```

## 2.7 Min Cost Max Flow

```

39 }
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

// Min Cost Max Flow (brunomaletta)
//
// min_cost_flow(s, t, f) computa o par (fluxo, custo)
// com max(flujo) <= f que tenha min(custo)
// min_cost_flow(s, t) -> Fluxo maximo de custo
// minimo de s pra t
// Se for um dag, da pra substituir o SPFA por uma DP
// pra nao
// pagar O(nm) no comeco
// Se nao tiver aresta com custo negativo, nao
// precisa do SPFA
//
// O(nm + f * m log n)

template<typename T> struct mcmf {
    struct edge {
        int to, rev, flow, cap; // para, id da
        reversa, fluxo, capacidade
        bool res; // se eh reversa
        T cost; // custo da unidade de fluxo
        edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false) {}
        edge(int to_, int rev_, int flow_, int cap_,
            T cost_, bool res_)
            : to(to_), rev(rev_), flow(flow_), cap(
            cap_), res(res_), cost(cost_) {}
    };

    vector<vector<edge>> g;
    vector<int> par_idx, par;
    T inf;
    vector<T> dist;

    mcmf(int n) : g(n), par_idx(n), par(n), inf(
        numeric_limits<T>::max()/3) {}

    void add(int u, int v, int w, T cost) { // de u
        pra v com cap w e custo cost
        edge a = edge(v, g[v].size(), 0, w, cost,
            false);
        edge b = edge(u, g[u].size(), 0, 0, -cost,
            true);

        g[u].push_back(a);
        g[v].push_back(b);
    }

    vector<T> spfa(int s) { // nao precisa se nao
        tiver custo negativo
        deque<int> q;
        vector<bool> is_inside(g.size(), 0);
        dist = vector<T>(g.size(), inf);

        dist[s] = 0;
        q.push_back(s);
        is_inside[s] = true;

        while (!q.empty()) {
            int v = q.front();
            q.pop_front();
            is_inside[v] = false;

            for (int i = 0; i < g[v].size(); i++) {
                auto [to, rev, flow, cap, res, cost]
                = g[v][i];
                if (flow < cap and dist[v] + cost <
                    dist[to]) {
                    dist[to] = dist[v] + cost;
                    if (is_inside[to]) continue;
                    if (!q.empty() and dist[to] >
                        dist[q.front()]) q.push_back(to);
                    else q.push_front(to);
                    is_inside[to] = true;
                }
            }
        }
        return dist;
    }

    bool dijkstra(int s, int t, vector<T>& pot) {
        priority_queue<pair<T, int>, vector<pair<T,
        int>>, greater<>> q;
        dist = vector<T>(g.size(), inf);
        dist[s] = 0;
        q.emplace(0, s);
        while (q.size()) {
            auto [d, v] = q.top();
            q.pop();
            if (dist[v] < d) continue;
            for (int i = 0; i < g[v].size(); i++) {
                auto [to, rev, flow, cap, res, cost]
                = g[v][i];
                cost += pot[v] - pot[to];
                if (flow < cap and dist[v] + cost <
                    dist[to]) {
                    dist[to] = dist[v] + cost;
                    q.emplace(dist[to], to);
                    par_idx[to] = i, par[to] = v;
                }
            }
        }
        return dist[t] < inf;
    }

    pair<int, T> min_cost_flow(int s, int t, int flow
    = INF) {
        vector<T> pot(g.size(), 0);
        pot = spfa(s); // mudar algoritmo de caminho
        minimo aqui

        int f = 0;
        T ret = 0;
        while (f < flow and dijkstra(s, t, pot)) {
            for (int i = 0; i < g.size(); i++)
                if (dist[i] < inf) pot[i] += dist[i];

            int mn_flow = flow - f, u = t;
            while (u != s) {
                mn_flow = min(mn_flow,
                    g[par[u]][par_idx[u]].cap - g[par
                    [u]][par_idx[u]].flow);
                u = par[u];
            }
            ret += pot[t] * mn_flow;

            u = t;
            while (u != s) {
                g[par[u]][par_idx[u]].flow += mn_flow
                ;
                g[u][g[par[u]][par_idx[u]].rev].flow
                -= mn_flow;
                u = par[u];
            }
            f += mn_flow;
        }
        return make_pair(f, ret);
    }
}

```

```

119 // Opcional: retorna as arestas originais por
120 // onde passa flow = cap
121 vector<pair<int,int>> recover() {
122     vector<pair<int,int>> used;
123     for (int i = 0; i < g.size(); i++) for (edge
124         e : g[i])
125         if (e.flow == e.cap && !e.res) used.
126     push_back({i, e.to});
127     return used;
128 }

```

## 2.8 Has Negative Cycle

```

1 // Edson
2
3 using edge = tuple<int, int, int>;
4
5 bool has_negative_cycle(int s, int N, const vector<
6     edge>& edges)
7 {
8     const int INF { 1e9+17 };
9
10    vector<int> dist(N + 1, INF);
11    dist[s] = 0;
12
13    for (int i = 1; i <= N - 1; i++) {
14        for (auto [u, v, w] : edges) {
15            if (dist[u] < INF && dist[v] > dist[u] +
16                w) {
17                dist[v] = dist[u] + w;
18            }
19        }
20    }
21
22    for (auto [u, v, w] : edges) {
23        if (dist[u] < INF && dist[v] > dist[u] + w) {
24            return true;
25        }
26    }
27
28    return false;
29 }

```

## 2.9 Dinic

```

1 // Dinic / Dinitz
2 //
3 // max-flow / min-cut
4 //
5 // https://cses.fi/problemset/task/1694/
6 //
7 // O(E * V^2)
8
9 using ll = long long;
10 const ll FLOW_INF = 1e18 + 7;
11
12 struct Edge {
13     int from, to;
14     ll cap, flow;
15     Edge* residual; // a inversa da minha aresta
16
17     Edge() {}
18
19     Edge(int from, int to, ll cap) : from(from), to(
20         to), cap(cap), flow(0) {}
21
22     ll remaining_cap() {
23         return cap - flow;
24     }
25
26     void augment(ll bottle_neck) {

```

```

26         flow += bottle_neck;
27         residual->flow -= bottle_neck;
28     }
29
30     bool is_residual() {
31         return cap == 0;
32     }
33 };
34
35 struct Dinic {
36     int n;
37     vector<vector<Edge*>> adj;
38     vector<int> level, next;
39
40     Dinic(int n): n(n) {
41         adj.assign(n+1, vector<Edge*>());
42         level.assign(n+1, -1);
43         next.assign(n+1, 0);
44     }
45
46     void add_edge(int from, int to, ll cap) {
47         auto e1 = new Edge(from, to, cap);
48         auto e2 = new Edge(to, from, 0);
49
50         e1->residual = e2;
51         e2->residual = e1;
52
53         adj[from].push_back(e1);
54         adj[to].push_back(e2);
55     }
56
57     bool bfs(int s, int t) {
58         fill(level.begin(), level.end(), -1);
59         queue<int> q;
60
61         q.push(s);
62         level[s] = 1;
63
64         while (q.size()) {
65             int curr = q.front();
66             q.pop();
67
68             for (auto edge : adj[curr]) {
69                 if (edge->remaining_cap() > 0 &&
70                     level[edge->to] == -1) {
71                     level[edge->to] = level[curr] +
72                         1;
73                     q.push(edge->to);
74                 }
75             }
76         }
77
78         return level[t] != -1;
79     }
80
81     ll dfs(int x, int t, ll flow) {
82         if (x == t) return flow;
83
84         for (int& cid = next[x]; cid < (int)adj[x].
85             size(); cid++) {
86             auto& edge = adj[x][cid];
87             ll cap = edge->remaining_cap();
88
89             if (cap > 0 && level[edge->to] == level[x
90                 ] + 1) {
91                 ll sent = dfs(edge->to, t, min(flow,
92                     cap)); // bottle neck
93                 if (sent > 0) {
94                     edge->augment(sent);
95                     return sent;
96                 }
97             }
98         }
99     }

```



```

94         return 0;
95     }
96 }
97
98 ll solve(int s, int t) {
99     ll max_flow = 0;
100
101     while (bfs(s, t)) {
102         fill(next.begin(), next.end(), 0);
103
104         while (ll sent = dfs(s, t, FLOW_INF)) {
105             max_flow += sent;
106         }
107     }
108
109     return max_flow;
110 }
111
112 // path recover
113 vector<bool> vis;
114 vector<int> curr;
115
116 bool dfs2(int x, int& t) {
117     vis[x] = true;
118     bool arrived = false;
119
120     if (x == t) {
121         curr.push_back(x);
122         return true;
123     }
124
125     for (auto e : adj[x]) {
126         if (e->flow > 0 && !vis[e->to]) { // !e->
127             is_residual() &&
128             bool aux = dfs2(e->to, t);
129
130             if (aux) {
131                 arrived = true;
132                 e->flow--;
133             }
134         }
135     }
136
137     if (arrived) curr.push_back(x);
138
139     return arrived;
140 }
141
142 vector<vector<int>> get_paths(int s, int t) {
143     vector<vector<int>> ans;
144
145     while (true) {
146         curr.clear();
147         vis.assign(n+1, false);
148
149         if (!dfs2(s, t)) break;
150
151         reverse(curr.begin(), curr.end());
152         ans.push_back(curr);
153     }
154
155     return ans;
156 };

```

## 3 String

### 3.1 Trie Xor

```

1 // TrieXOR
2 //

```

```

3 // adiciona, remove e verifica se existe strings
4 // binarias
5 // max_xor(x) = maximiza o xor de x com algum valor
6 // da trie
7 //
8 // raiz = 0
9 //
10 // https://codeforces.com/problemset/problem/706/D
11 // 0(|s|) adicionar, remover e buscar
12
13 struct TrieXOR {
14     int n, alph_sz, nxt;
15     vector<vector<int>> trie;
16     vector<int> finish, paths;
17
18     TrieXOR() {}
19
20     TrieXOR(int n, int alph_sz = 2) : n(n), alph_sz(
21         alph_sz) {
22         nxt = 1;
23         trie.assign(n, vector<int>(alph_sz));
24         finish.assign(n * alph_sz, 0);
25         paths.assign(n * alph_sz, 0);
26     }
27
28     void add(int x) {
29         int curr = 0;
30
31         for (int i = 31; i >= 0; i--) {
32             int b = ((x << i) > 0);
33
34             if (trie[curr][b] == 0)
35                 trie[curr][b] = nxt++;
36
37             paths[curr]++;
38             curr = trie[curr][b];
39         }
40
41         paths[curr]++;
42         finish[curr]++;
43     }
44
45     void rem(int x) {
46         int curr = 0;
47
48         for (int i = 31; i >= 0; i--) {
49             int b = ((x << i) > 0);
50
51             paths[curr]--;
52             curr = trie[curr][b];
53         }
54
55         paths[curr]--;
56         finish[curr]--;
57     }
58
59     int search(int x) {
60         int curr = 0;
61
62         for (int i = 31; i >= 0; i--) {
63             int b = ((x << i) > 0);
64
65             if (trie[curr][b] == 0) return false;
66
67             curr = trie[curr][b];
68         }
69
70         return (finish[curr] > 0);
71     }
72
73     int max_xor(int x) { // maximum xor with x and
74         any number of trie

```

```

72     int curr = 0, ans = 0;
73
74     for (int i = 31; i >= 0; i--) {
75         int b = ((x & (1 << i)) > 0);
76         int want = b ^ 1;
77
78         if (trie[curr][want] == 0 || paths[trie[
curr][want]] == 0) want ^= 1;
79         if (trie[curr][want] == 0 || paths[trie[
curr][want]] == 0) break;
80         if (want != b) ans |= (1 << i);
81
82         curr = trie[curr][want];
83     }
84
85     return ans;
86 }
87 };

```

### 3.2 Split

```

1 vector<string> split(string s, char key=' ') {
2     vector<string> ans;
3     string aux = "";
4
5     for (int i = 0; i < (int)s.size(); i++) {
6         if (s[i] == key) {
7             if (aux.size() > 0) {
8                 ans.push_back(aux);
9                 aux = "";
10            }
11            } else {
12                aux += s[i];
13            }
14        }
15
16        if ((int)aux.size() > 0) {
17            ans.push_back(aux);
18        }
19
20        return ans;
21 }

```

### 3.3 Is Substring

```

1 // equivalente ao in do python
2
3 bool is_substring(string a, string b){ // verifica se
4     a Ã substring de b
5     for(int i = 0; i < b.size(); i++){
6         int it = i, jt = 0; // b[it], a[jt]
7
8         while(it < b.size() && jt < a.size()){
9             if(b[it] != a[jt])
10                break;
11
12            it++;
13            jt++;
14
15            if(jt == a.size())
16                return true;
17        }
18
19        return false;
20 }

```

### 3.4 Hash

```

1 struct Hash {
2     ll MOD, P;
3     int n; string s;

```

```

4     vector<ll> h, hi, p;
5     Hash() {}
6     Hash(string s, ll MOD, ll P = 31): s(s), MOD(MOD)
, P(P), n(s.size()), h(n), hi(n), p(n) {
7         for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1)
% MOD;
8         for (int i=0;i<n;i++)
9             h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
10        for (int i=n-1;i>=0;i--)
11            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
% MOD;
12    }
13    int query(int l, int r) {
14        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]:0)%MOD :
0));
15        return hash < 0 ? hash + MOD : hash;
16    }
17    int query_inv(int l, int r) {
18        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
+1] % MOD : 0));
19        return hash < 0 ? hash + MOD : hash;
20    }
21 };
22
23 struct DoubleHash {
24     const ll MOD1 = 90264469;
25     const ll MOD2 = 25699183;
26
27     Hash hash1, hash2;
28
29     DoubleHash();
30
31     DoubleHash(string s) : hash1(s, MOD1), hash2(s,
MOD2) {}
32
33     pair<int, int> query(int l, int r) {
34         return { hash1.query(l, r), hash2.query(l, r)
};
35     }
36
37     pair<int, int> query_inv(int l, int r) {
38         return { hash1.query_inv(l, r), hash2.
query_inv(l, r) };
39     }
40 };
41
42 struct TripleHash {
43     const ll MOD1 = 90264469;
44     const ll MOD2 = 25699183;
45     const ll MOD3 = 81249169;
46
47     Hash hash1, hash2, hash3;
48
49     TripleHash();
50
51     TripleHash(string s) : hash1(s, MOD1), hash2(s,
MOD2), hash3(s, MOD3) {}
52
53     tuple<int, int, int> query(int l, int r) {
54         return { hash1.query(l, r), hash2.query(l, r)
, hash3.query(l, r) };
55     }
56
57     tuple<int, int, int> query_inv(int l, int r) {
58         return { hash1.query_inv(l, r), hash2.
query_inv(l, r), hash3.query_inv(l, r) };
59     }
60 };
61
62 struct HashK {
63     vector<ll> primes; // more primes = more hashes
64     vector<Hash> hash;
65

```

```

66     HashK();
67
68     HashK(string s, vector<ll> primes): primes(primes)
69     {
70         for (auto p : primes) {
71             hash.push_back(Hash(s, p));
72         }
73
74         vector<int> query(int l, int r) {
75             vector<int> ans;
76
77             for (auto h : hash) {
78                 ans.push_back(h.query(l, r));
79             }
80
81             return ans;
82         }
83
84         vector<int> query_inv(int l, int r) {
85             vector<int> ans;
86
87             for (auto h : hash) {
88                 ans.push_back(h.query_inv(l, r));
89             }
90
91             return ans;
92         }
93 };

```

## 4 Math

### 4.1 Fft Quirino

```

1 // FFT
2 //
3 // boa em memória e ok em tempo
4 //
5 // https://codeforces.com/group/YgJmumGtHD/contest
6 // 528947/problem/H (maratona mineira)
7
8 using cd = complex<double>;
9 const double PI = acos(-1);
10
11 void fft(vector<cd> &A, bool invert) {
12     int N = size(A);
13
14     for (int i = 1, j = 0; i < N; i++) {
15         int bit = N >> 1;
16         for (; j & bit; bit >>= 1)
17             j ^= bit;
18
19         if (i < j)
20             swap(A[i], A[j]);
21     }
22
23     for (int len = 2; len <= N; len <= 1) {
24         double ang = 2 * PI / len * (invert ? -1 : 1);
25         cd wlen(cos(ang), sin(ang));
26         for (int i = 0; i < N; i += len) {
27             cd w(1);
28             for (int j = 0; j < len/2; j++) {
29                 cd u = A[i+j], v = A[i+j+len/2] * w;
30                 A[i+j] = u + v;
31                 A[i+j+len/2] = u - v;
32                 w *= wlen;
33             }
34         }
35     }
36
37     if (invert) {

```

```

38         for (auto &x : A)
39             x /= N;
40     }
41 }
42
43 vector<int> multiply(vector<int> const& A, vector<int>
44 > const& B) {
45     vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
46
47     int N = 1;
48     while (N < size(A) + size(B))
49         N <= 1;
50     fa.resize(N);
51     fb.resize(N);
52
53     fft(fa, false);
54     fft(fb, false);
55     for (int i = 0; i < N; i++)
56         fa[i] *= fb[i];
57     fft(fa, true);
58
59     vector<int> result(N);
60     for (int i = 0; i < N; i++)
61         result[i] = round(fa[i].real());
62     return result;
63 }

```

### 4.2 Ceil

```

1 using ll = long long;
2
3 // avoid overflow
4 ll division_ceil(ll a, ll b) {
5     return 1 + ((a - 1) / b); // if a != 0
6 }
7
8 int intceil(int a, int b) {
9     return (a+b-1)/b;
10 }

```

### 4.3 Division Trick

```

1 for(int l = 1, r; l <= n; l = r + 1) {
2     r = n / (n / l);
3     // n / x yields the same value for l <= x <= r
4 }
5 for(int l, r = n; r > 0; r = l - 1) {
6     int tmp = (n + r - 1) / r;
7     l = (n + tmp - 1) / tmp;
8     // (n+x-1) / x yields the same value for l <= x
9     <= r
10 }

```

### 4.4 Fexp

```

1 using ll = long long;
2
3 ll fexp(ll base, ll exp, ll m) {
4     ll ans = 1;
5     base %= m;
6
7     while (exp > 0) {
8         if (exp % 2 == 1) {
9             ans = (ans * base) % m;
10        }
11
12        base = (base * base) % m;
13        exp /= 2;
14    }
15
16    return ans;
17 }

```

## 4.5 Sieve

```

1 // nao "otimizado"
2
3 vector<bool> sieve(int lim=1e5+17) {
4     vector<bool> isprime(lim+1, true);
5
6     isprime[0] = isprime[1] = false;
7
8     for (int i = 2; i*i < lim; i++) {
9         if (isprime[i]) {
10             for (int j = i+i; j < lim; j += i) {
11                 isprime[j] = false;
12             }
13         }
14     }
15
16     return isprime;
17 }

```

## 4.6 Divisors

```

1 vector<ll> divisors(ll n) {
2     vector<ll> ans;
3
4     for (ll i = 1; i*i <= n; i++) {
5         if (n%i == 0) {
6             ll value = n/i;
7
8             ans.push_back(i);
9             if (value != i) {
10                 ans.push_back(value);
11             }
12         }
13     }
14
15     return ans;
16 }

```

## 4.7 Log Any Base

```

1 int intlog(double base, double x) {
2     return (int)(log(x) / log(base));
3 }

```

## 4.8 Generate Primes

```

1 // crivo nao otimizado
2
3 vector<int> generate_primes(int lim=1e5+17) {
4     vector<int> primes;
5     vector<bool> isprime(lim+1, true);
6
7     isprime[0] = isprime[1] = false;
8
9     for (int i = 2; i*i < lim; i++) {
10         if (isprime[i]) {
11             primes.push_back(i);
12
13             for (int j = i+i; j < lim; j += i) {
14                 isprime[j] = false;
15             }
16         }
17     }
18
19     return primes;
20 }

```

## 4.9 Factorization

```

1 // nson
2
3 using ll = long long;
4
5 vector<pair<ll, int>> factorization(ll n) {
6     vector<pair<ll, int>> ans;
7
8     for (ll p = 2; p*p <= n; p++) {
9         if (n%p == 0) {
10             int expoente = 0;
11
12             while (n%p == 0) {
13                 n /= p;
14                 expoente++;
15             }
16
17             ans.push_back({p, expoente});
18         }
19     }
20
21     if (n > 1) {
22         ans.push_back({n, 1});
23     }
24
25     return ans;
26 }

```

## 4.10 Is Prime

```

1 bool is_prime(ll n) {
2     if (n <= 1) return false;
3     if (n == 2) return true;
4
5     for (ll i = 2; i*i <= n; i++) {
6         if (n % i == 0)
7             return false;
8     }
9
10    return true;
11 }

```

# 5 Geometry

## 5.1 Convex Hull

```

1 // Convex Hull - Monotone Chain
2 //
3 // Convex Hull is the subset of points that forms the
4 // smallest convex polygon
5 // which encloses all points in the set.
6 // https://cses.fi/problemset/task/2195/
7 // https://open.kattis.com/problems/convexhull (
8 // counterclockwise)
9 //
10 // O(n log(n))
11
12 typedef long long ftype;
13
14 struct Point {
15     ftype x, y;
16
17     Point() {}
18     Point(ftype x, ftype y) : x(x), y(y) {}
19
20     bool operator<(Point o) {
21         if (x == o.x) return y < o.y;
22         return x < o.x;
23     }
24
25     bool operator==(Point o) {

```

```

25     return x == o.x && y == o.y;
26 }
27 };
28
29 ftype cross(Point a, Point b, Point c) {
30     // v: a -> c
31     // w: a -> b
32
33     // v: c.x - a.x, c.y - a.y
34     // w: b.x - a.x, b.y - a.y
35
36     return (c.x - a.x) * (b.y - a.y) - (c.y - a.y) *
37         (b.x - a.x);
38 }
39
40 ftype dir(Point a, Point b, Point c) {
41     // 0 -> colineares
42     // -1 -> esquerda
43     // 1 -> direita
44
45     ftype cp = cross(a, b, c);
46
47     if (cp == 0) return 0;
48     else if (cp < 0) return -1;
49     else return 1;
50 }
51
52 vector<Point> convex_hull(vector<Point> points) {
53     sort(points.begin(), points.end());
54     points.erase( unique(points.begin(), points.end())
55         ), points.end()); // somente pontos distintos
56     int n = points.size();
57
58     if (n == 1) return { points[0] };
59
60     vector<Point> upper_hull = {points[0], points
61         [1]};
62     for (int i = 2; i < n; i++) {
63         upper_hull.push_back(points[i]);
64
65         int sz = upper_hull.size();
66
67         while (sz >= 3 && dir(upper_hull[sz-3],
68             upper_hull[sz-2], upper_hull[sz-1]) == -1) {
69             upper_hull.pop_back();
70             upper_hull.pop_back();
71             upper_hull.push_back(points[i]);
72             sz--;
73         }
74     }
75
76     vector<Point> lower_hull = {points[n-1], points[n
77         -2]};
78     for (int i = n-3; i >= 0; i--) {
79         lower_hull.push_back(points[i]);
80
81         int sz = lower_hull.size();
82
83         while (sz >= 3 && dir(lower_hull[sz-3],
84             lower_hull[sz-2], lower_hull[sz-1]) == -1) {
85             lower_hull.pop_back();
86             lower_hull.pop_back();
87             lower_hull.push_back(points[i]);
88             sz--;
89         }
90     }
91
92     // reverse(lower_hull.begin(), lower_hull.end());
93     // counterclockwise
94
95     for (int i = (int)lower_hull.size() - 2; i > 0; i
96         --) {
97         upper_hull.push_back(lower_hull[i]);
98     }
99 }

```

```

90 }
91
92 return upper_hull;
93 }

```

## 6 DP

### 6.1 Edit Distance

```

1 // Edit Distance / Levenshtein Distance
2 //
3 // numero minimo de operacoes
4 // para transformar
5 // uma string em outra
6 //
7 // tamanho da matriz da dp eh |a| x |b|
8 // edit_distance(a.size(), b.size(), a, b)
9 //
10 // https://cses.fi/problemset/task/1639
11 //
12 // 0(n^2)
13
14 int tb[MAX][MAX];
15
16 int edit_distance(int i, int j, string &a, string &b)
17 {
18     if (i == 0) return j;
19     if (j == 0) return i;
20
21     int &ans = tb[i][j];
22
23     if (ans != -1) return ans;
24
25     ans = min({
26         edit_distance(i-1, j, a, b) + 1,
27         edit_distance(i, j-1, a, b) + 1,
28         edit_distance(i-1, j-1, a, b) + (a[i-1] != b[
29             j-1])
30     });
31
32     return ans;
33 }

```

### 6.2 Range Dp

```

1 // Range DP 1: https://codeforces.com/problemset/
2 // problem/1132/F
3 //
4 // You may apply some operations to this string
5 // in one operation you can delete some contiguous
6 // substring of this string
7 // if all letters in the substring you delete are
8 // equal
9 // calculate the minimum number of operations to
10 // delete the whole string s
11
12 #include <bits/stdc++.h>
13
14 using namespace std;
15
16 const int MAX = 510;
17
18 int n, tb[MAX][MAX];
19 string s;
20
21 int dp(int left, int right) {
22     if (left > right) return 0;
23
24     int& mem = tb[left][right];
25     if (mem != -1) return mem;
26
27     // ...
28 }

```

```

23     mem = 1 + dp(left+1, right); // gastar uma
    operaçãõ arrumando sã o cara atual
24     for (int i = left+1; i <= right; i++) {
25         if (s[left] == s[i]) {
26             mem = min(mem, dp(left+1, i-1) + dp(i,
    right));
27         }
28     }
29     return mem;
30 }
31
32 int main() {
33     ios::sync_with_stdio(false);
34     cin.tie(NULL);
35
36     cin >> n >> s;
37     memset(tb, -1, sizeof(tb));
38     cout << dp(0, n-1) << '\n';
39
40     return 0;
41 }
42

```

### 6.3 Lis Segtree

```

1  int n, arr[MAX], aux[MAX]; cin >> n;
2  for (int i = 0; i < n; i++) {
3      cin >> arr[i];
4      aux[i] = arr[i];
5  }
6
7  sort(aux, aux+n);
8
9  Segtree st(n); // seg of maximum
10
11 int ans = 0;
12 for (int i = 0; i < n; i++) {
13     int it = lower_bound(aux, aux+n, arr[i]) - aux;
14     int lis = st.query(0, it) + 1;
15
16     st.update(it, lis);
17
18     ans = max(ans, lis);
19 }
20
21 cout << ans << '\n';

```

### 6.4 Digit Dp 2

```

1  // Digit DP 2: https://cses.fi/problemset/task/2220
2  //
3  // Number of integers between a and b
4  // where no two adjacent digits are the same
5
6  #include <bits/stdc++.h>
7
8  using namespace std;
9  using ll = long long;
10
11 const int MAX = 20; // 10^18
12
13 ll tb[MAX][MAX][2][2];
14
15 ll dp(string& number, int pos, int last_digit, bool
    under, bool started) {
16     if (pos >= (int)number.size()) {
17         return 1;
18     }
19
20     ll& mem = tb[pos][last_digit][under][started];
21     if (mem != -1) return mem;
22     mem = 0;

```

```

23     int limit = 9;
24     if (!under) limit = number[pos] - '0';
25
26     for (int digit = 0; digit <= limit; digit++) {
27         if (started && digit == last_digit) continue;
28
29         bool is_under = under || (digit < limit);
30         bool is_started = started || (digit != 0);
31
32         mem += dp(number, pos+1, digit, is_under,
    is_started);
33     }
34
35     return mem;
36 }
37
38 ll solve(ll ubound) {
39     memset(tb, -1, sizeof(tb));
40     string number = to_string(ubound);
41     return dp(number, 0, 10, 0, 0);
42 }
43
44 int main() {
45     ios::sync_with_stdio(false);
46     cin.tie(NULL);
47
48     ll a, b; cin >> a >> b;
49     cout << solve(b) - solve(a-1) << '\n';
50
51     return 0;
52 }
53

```

### 6.5 Lis Binary Search

```

1  int lis(vector<int> arr) {
2      vector<int> dp;
3
4      for (auto e : arr) {
5          int pos = lower_bound(dp.begin(), dp.end(), e
    ) - dp.begin();
6
7          if (pos == (int)dp.size()) {
8              dp.push_back(e);
9          } else {
10             dp[pos] = e;
11         }
12     }
13
14     return (int)dp.size();
15 }

```

### 6.6 Lcs

```

1  // LCS (Longest Common Subsequence)
2  //
3  // maior subsequencia comum entre duas strings
4  //
5  // tamanho da matriz da dp eh |a| x |b|
6  // lcs(a, b) = string da melhor resposta
7  // dp[a.size()][b.size()] = tamanho da melhor
    resposta
8  //
9  // https://atcoder.jp/contests/dp/tasks/dp_f
10 //
11 // O(n^2)
12
13 string lcs(string a, string b) {
14     int n = a.size();
15     int m = b.size();
16
17     int dp[n+1][m+1];

```

```

18 pair<int, int> p[n+1][m+1];
19
20 memset(dp, 0, sizeof(dp));
21 memset(p, -1, sizeof(p));
22
23 for (int i = 1; i <= n; i++) {
24     for (int j = 1; j <= m; j++) {
25         if (a[i-1] == b[j-1]) {
26             dp[i][j] = dp[i-1][j-1] + 1;
27             p[i][j] = {i-1, j-1};
28         } else {
29             if (dp[i-1][j] > dp[i][j-1]) {
30                 dp[i][j] = dp[i-1][j];
31                 p[i][j] = {i-1, j};
32             } else {
33                 dp[i][j] = dp[i][j-1];
34                 p[i][j] = {i, j-1};
35             }
36         }
37     }
38 }
39
40 // recuperar resposta
41
42 string ans = "";
43 pair<int, int> curr = {n, m};
44
45 while (curr.first != 0 && curr.second != 0) {
46     auto [i, j] = curr;
47
48     if (a[i-1] == b[j-1]) {
49         ans += a[i-1];
50     }
51
52     curr = p[i][j];
53 }
54
55 reverse(ans.begin(), ans.end());
56
57 return ans;
58 }

```

## 6.7 Digit Dp

```

1 // Digit DP 1: https://atcoder.jp/contests/dp/tasks/
  dp_s
2 //
3 // find the number of integers between 1 and K (
  inclusive)
4 // where the sum of digits in base ten is a multiple
  of D
5
6 #include <bits/stdc++.h>
7
8 using namespace std;
9
10 const int MOD = 1e9+7;
11
12 string k;
13 int d;
14
15 int tb[10010][110][2];
16
17 int dp(int pos, int sum, bool under) {
18     if (pos >= k.size()) return sum == 0;
19
20     int& mem = tb[pos][sum][under];
21     if (mem != -1) return mem;
22     mem = 0;
23
24     int limit = 9;
25     if (!under) limit = k[pos] - '0';
26

```

```

27     for (int digit = 0; digit <= limit; digit++) {
28         mem += dp(pos+1, (sum + digit) % d, under | (
  digit < limit));
29         mem %= MOD;
30     }
31
32     return mem;
33 }
34
35 int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(NULL);
38
39     cin >> k >> d;
40
41     memset(tb, -1, sizeof(tb));
42
43     cout << (dp(0, 0, false) - 1 + MOD) % MOD << '\n'
  ;
44
45     return 0;
46 }

```

## 7 General

### 7.1 Mix Hash

```

1 // magic hash function using mix
2
3 using ull = unsigned long long;
4 ull mix(ull o){
5     o+=0x9e3779b97f4a7c15;
6     o=(o^(o>>30))*0xbf58476d1ce4e5b9;
7     o=(o^(o>>27))*0x94d049bb133111eb;
8     return o^(o>>31);
9 }
10 ull hash(pii a) {return mix(a.first ^ mix(a.second))
  ;}

```

### 7.2 Xor 1 To N

```

1 // XOR sum from 1 to N
2 ll xor_1_to_n(ll n) {
3     if (n % 4 == 0) {
4         return n;
5     } else if (n % 4 == 1) {
6         return 1;
7     } else if (n % 4 == 2) {
8         return n + 1;
9     }
10
11     return 0;
12 }

```

### 7.3 Base Converter

```

1 const string digits = "0123456789
  ABCDEFGHIJKLMNOPQRSTUVWXYZ";
2
3 ll tobase10(string number, int base) {
4     map<char, int> val;
5     for (int i = 0; i < digits.size(); i++) {
6         val[digits[i]] = i;
7     }
8
9     ll ans = 0, pot = 1;
10
11     for (int i = number.size() - 1; i >= 0; i--) {
12         ans += val[number[i]] * pot;
13         pot *= base;
14     }

```

```

15
16     return ans;
17 }
18
19 string frombase10(ll number, int base) {
20     if (number == 0) return "0";
21
22     string ans = "";
23
24     while (number > 0) {
25         ans += digits[number % base];
26         number /= base;
27     }
28
29     reverse(ans.begin(), ans.end());
30
31     return ans;
32 }
33
34 // verifica se um número está na base especificada
35 bool verify_base(string num, int base) {
36     map<char, int> val;
37     for (int i = 0; i < digits.size(); i++) {
38         val[digits[i]] = i;
39     }
40
41     for (auto digit : num) {
42         if (val[digit] >= base) {
43             return false;
44         }
45     }
46
47     return true;
48 }

```

## 7.4 Min Priority Queue

```

1 template<class T> using min_priority_queue =
  priority_queue<T, vector<T>, greater<T>>;

```

## 7.5 Input By File

```

1 freopen("file.in", "r", stdin);
2 freopen("file.out", "w", stdout);

```

## 7.6 Template

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main() {
6     ios::sync_with_stdio(false);
7     cin.tie(NULL);
8
9
10
11     return 0;
12 }

```

## 7.7 First True

```

1 // Binary Search (first_true)
2 //
3 // first_true(2, 10, [](int x) { return x * x >= 30;
4 // }); // outputs 6
5 //
6 // [1, r]
7 // if none of the values in the range work, return hi
  + 1

```

```

8 //
9 // f(4) = false
10 // f(5) = false
11 // f(6) = true
12 // f(7) = true
13
14 int first_true(int lo, int hi, function<bool(int)> f)
15 {
16     hi++;
17     while (lo < hi) {
18         int mid = lo + (hi - lo) / 2;
19
20         if (f(mid)) {
21             hi = mid;
22         } else {
23             lo = mid + 1;
24         }
25     }
26     return lo;
27 }

```

## 7.8 Get Subsets Sum Iterative

```

1 vector<ll> get_subset_sums(int l, int r, vector<ll>&
  arr) {
2     vector<ll> ans;
3
4     int len = r-l+1;
5     for (int i = 0; i < (1 << len); i++) {
6         ll sum = 0;
7
8         for (int j = 0; j < len; j++) {
9             if (i & (1 << j)) {
10                 sum += arr[l + j];
11             }
12         }
13
14         ans.push_back(sum);
15     }
16
17     return ans;
18 }

```

## 7.9 Last True

```

1 // Binary Search (last_true)
2
3 // last_true(2, 10, [](int x) { return x * x <= 30;
4 // }); // outputs 5
5 //
6 // [1, r]
7 // if none of the values in the range work, return lo
  - 1
8 //
9 // f(1) = true
10 // f(2) = true
11 // f(3) = true
12 // f(4) = true
13 // f(5) = true
14 // f(6) = false
15 // f(7) = false
16 // f(8) = false
17 //
18 // last_true(1, 8, f) = 5
19 // last_true(7, 8, f) = 6
20
21 int last_true(int lo, int hi, function<bool(int)> f)
22 {
23     lo--;
24     while (lo < hi) {
25         int mid = lo + (hi - lo + 1) / 2;

```



```

25         if (f(mid)) {
26             lo = mid;
27         } else {
28             hi = mid - 1;
29         }
30     }
31     return lo;
32 }
33 }

```

## 7.10 Interactive

```

1 // you should use cout.flush() every cout
2 int query(int a) {
3     cout << "? " << a << '\n';
4     cout.flush();
5     char res; cin >> res;
6     return res;
7 }
8
9 // using endl you don't need
10 int query(int a) {
11     cout << "? " << a << endl;
12     char res; cin >> res;
13     return res;
14 }

```

## 7.11 Next Permutation

```

1 // output: 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2; 3,2,1;
2
3 vector<int> arr = {1, 2, 3};
4 int n = arr.size();
5
6 do {
7     for (auto e : arr) {
8         cout << e << ' ';
9     }
10    cout << '\n';
11 } while (next_permutation(arr.begin(), arr.end()));

```

## 7.12 Random

```

1 random_device dev;
2 mt19937 rng(dev());
3
4 uniform_int_distribution<mt19937::result_type> dist
5     (1, 6); // distribution in range [1, 6]
6 int val = dist(rng);

```

## 8 Primitives