

Competitive Programming Notebook

As Meninas Superpoderosas

Contents

1 DS	2	6 DP	15
1.1 Ordered Set	2	6.1 Edit Distance	15
1.2 Range Color Update	2	6.2 Range Dp	15
1.3 Bigk	2	6.3 Lis Segtree	16
1.4 Mex	3	6.4 Digit Dp 2	16
1.5 Segtree Lazy Iterative	3	6.5 Lis Binary Search	16
1.6 Cht	4	6.6 Lcs	16
1.7 Kruskal	4	6.7 Digit Dp	17
1.8 Dsu	4	7 General	17
2 Graph	5	7.1 Mix Hash	17
2.1 Dijkstra	5	7.2 Xor 1 To N	17
2.2 Ford Fulkerson	5	7.3 Base Converter	17
2.3 2sat	5	7.4 Min Priority Queue	18
2.4 Floyd Warshall	6	7.5 Flags	18
2.5 Lca	6	7.6 Input By File	18
2.6 Bfs	7	7.7 Template	18
2.7 Min Cost Max Flow	7	7.8 First True	18
2.8 Has Negative Cycle	8	7.9 Get Subsets Sum Iterative	18
2.9 3sat	9	7.10 Last True	18
2.10 Dinic	9	7.11 Overflow	19
3 String	11	7.12 Interactive	19
3.1 Trie Xor	11	7.13 Next Permutation	19
3.2 Split	11	7.14 Random	19
3.3 Is Substring	11	8 Primitives	19
3.4 Hash	12		
4 Math	12		
4.1 Fft Quirino	12		
4.2 Ceil	13		
4.3 Division Trick	13		
4.4 Fexp	13		
4.5 Sieve	13		
4.6 Divisors	13		
4.7 Log Any Base	13		
4.8 Generate Primes	13		
4.9 Factorization	14		
4.10 Is Prime	14		
4.11 Number Sum Product Of Divisors	14		
5 Geometry	14		
5.1 Convex Hull	14		

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
    // smaller than k -> int
11 // find_by_order(k) // k-th element in a set (
    // counting from zero) -> iterator
12 //
13 // https://cses.fi/problemset/task/2169
14 //
15 // O(log N) para insert, erase (com iterator),
    // order_of_key, find_by_order
16
17 using namespace __gnu_pbds;
18 template <typename T>
19 using ordered_set = tree<T,null_type,less<T>,
    rb_tree_tag,tree_order_statistics_node_update>;
20
21 void erase(ordered_set& a, int x){
22     int r = a.order_of_key(x);
23     auto it = a.find_by_order(r);
24     a.erase(it);
25 }

```

1.2 Range Color Update

```

1 // Range color update (brunomaletta)
2 //
3 // update(l, r, c) colore o range [l, r] com a cor c,
4 // e retorna os ranges que foram coloridos {l, r, cor}
5 //
6 // query(i) retorna a cor da posicao i
7 //
8 // Complexidades (para q operacoes):
9 // update - O(log(q)) amortizado
10 // query - O(log(q))
11
12 template<typename T> struct color {
13     set<tuple<int, int, T>> se;
14
15     vector<tuple<int, int, T>> update(int l, int r, T
        val) {
16         auto it = se.upper_bound({r, INF, val});
17         if (it != se.begin() and get<1>(*prev(it)) >
            r) {
18             auto [L, R, V] = *--it;
19             se.erase(it);
20             se.emplace(L, r, V), se.emplace(r+1, R, V);
21         }
22         it = se.lower_bound({l, -INF, val});
23         if (it != se.begin() and get<1>(*prev(it)) >=
            l) {
24             auto [L, R, V] = *--it;
25             se.erase(it);
26             se.emplace(L, l-1, V), it = se.emplace(l,
                R, V).first;
27         }
28         vector<tuple<int, int, T>> ret;
29         for (; it != se.end() and get<0>(*it) <= r;
            it = se.erase(it))
30             ret.push_back(*it);
31     }
32 };

```

```

30         se.emplace(l, r, val);
31         return ret;
32     }
33     T query(int i) {
34         auto it = se.upper_bound({i, INF, T()});
35         if (it == se.begin() or get<1>(*--it) < i)
36             return -1; // nao tem
37         return get<2>(*it);
38     }
39 };

```

1.3 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 };

```

```

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 };

```

1.4 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 };

```

1.5 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-1)/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[
esq], tam/2);
92         }
93     }
94 }
95
96 int findkth(ll k){
97     // kth smallest, 0(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 };

```

1.6 Cht

```

1 // CHT (tiagodfs)
2
3 const ll is_query = -LLINF;
4 struct Line{
5     ll m, b;
6     mutable function<const Line*> succ;
7     bool operator<(const Line& rhs) const{
8         if(rhs.b != is_query) return m < rhs.m;
9         const Line* s = succ();
10        if(!s) return 0;
11        ll x = rhs.m;
12        return b - s->b < (s->m - m) * x;
13    }
14 };
15 struct Cht : public multiset<Line>{ // maintain max m
16     *x+b
17     bool bad(iterator y){
18         auto z = next(y);
19         if(y == begin()){
20             if(z == end()) return 0;
21             return y->m == z->m && y->b <= z->b;
22         }
23         auto x = prev(y);
24         if(z == end()) return y->m == x->m && y->b <=
x->b;
25         return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)
(y->b - z->b)*(y->m - x->m);
26     }
27     void insert_line(ll m, ll b){ // min -> insert (-
m,-b) -> -eval()
28         auto y = insert({ m, b });
29         y->succ = [=]{ return next(y) == end() ? 0 :
&*next(y); };
30         if(bad(y)){ erase(y); return; }
31         while(next(y) != end() && bad(next(y))) erase
(next(y));
32         while(y != begin() && bad(prev(y))) erase(
prev(y));
33     }
34     ll eval(ll x){
35         auto l = *lower_bound((Line) { x, is_query })
36         ;
37         return l.m * x + l.b;
38     }
39 };

```

1.7 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),
weight(weight) {}
8
9     bool operator<(Edge const& other) {
10         return weight < other.weight;
11     }
12 };
13
14 vector<Edge> kruskal(vector<Edge> edges, int n) {
15     vector<Edge> result;
16     ll cost = 0;
17
18     sort(edges.begin(), edges.end());
19     DSU dsu(n);
20
21     for (auto e : edges) {
22         if (!dsu.same(e.u, e.v)) {
23             cost += e.weight;
24             result.push_back(e);
25             dsu.unite(e.u, e.v);
26         }
27     }
28
29     return result;
30 }

```

1.8 Dsu

```

1 // DSU
2 //
3 // https://judge.yosupo.jp/submission/126864
4
5 struct DSU {
6     int n = 0, components = 0;
7     vector<int> parent;
8     vector<int> size;
9
10    DSU(int nn){
11        n = nn;
12        components = n;
13        size.assign(n + 5, 1);
14        parent.assign(n + 5, 0);
15        iota(parent.begin(), parent.end(), 0);
16    }
17
18    int find(int x){
19        if(x == parent[x]) {
20            return x;
21        }
22        //path compression
23        return parent[x] = find(parent[x]);
24    }
25
26    void join(int a, int b){
27        a = find(a);
28        b = find(b);
29
30        if(a == b) {
31            return;
32        }
33
34        if(size[a] < size[b]) {
35            swap(a, b);
36        }
37
38        parent[b] = a;
39        size[a] += size[b];

```

```

40     components -= 1;
41 }
42
43 int sameSet(int a, int b) {
44     a = find(a);
45     b = find(b);
46     return a == b;
47 }
48 };

```

2 Graph

2.1 Dijkstra

```

1  const int INF = 1e9+17;
2  vector<vector<pair<int, int>>> adj; // {neighbor,
   weight}
3
4  void dijkstra(int s, vector<int> & d, vector<int> & p
   ) {
5      int n = adj.size();
6      d.assign(n, INF);
7      p.assign(n, -1);
8
9      d[s] = 0;
10     set<pair<int, int>> q;
11     q.insert({0, s});
12     while (!q.empty()) {
13         int v = q.begin()->second;
14         q.erase(q.begin());
15
16         for (auto edge : adj[v]) {
17             int to = edge.first;
18             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 não
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     return 0;
45 }
46
47 ll max_flow(int s, int t) { // source and sink
48     ll total = 0;
49     ll sent = -1;
50
51     while (sent != 0) {
52         memset(used, 0, sizeof(used));
53         sent = dfs(s, t, INT_MAX);
54         total += sent;
55     }
56
57     return total;
58 }
59 };

```

2.3 2sat

```

1  // 2SAT
2  //
3  // verifica se existe e encontra solu
4  // para fórmulas 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)
   ^1 = 4
10
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 não
   foram testadas)
14 //
15 // O(n + m)
16
17 struct sat {
18     int n, tot;
19     vector<vector<int>> adj, adjt; // grafo original,
   grafo transposto
20     vector<int> vis, comp, ans;
21     stack<int> topo; // ordem topológica
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     }
104 }
105
106 pair<bool, vector<int>> solve() {
107     ans.assign(n, -1);
108     comp.assign(2*tot, -1);
109     vis.assign(2*tot, 0);
110     int id = 1;
111
112     for (int i = 0; i < 2*tot; i++) if (!vis[i])
113         dfs(i);
114
115     vis.assign(2*tot, 0);
116     while (topo.size()) {
117         auto x = topo.top();
118         topo.pop();
119
120         if (!vis[x]) {
121             dfst(x, id);
122             id++;
123         }
124     }
125
126     for (int i = 0; i < tot; i++) {
127         if (comp[2*i] == comp[2*i+1]) return {
128             false, {} };
129         ans[i] = (comp[2*i] > comp[2*i+1]);
130     }
131
132     return {true, ans};
133 }
134 };

```

2.4 Floyd Warshall

```

1 const long long LLINF = 0x3f3f3f3f3f3f3f3fLL;
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     }
28 }

```

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     }
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         return up[a][0];
62     }
63
64     int dist(int a, int b) {
65         return dep[a] + dep[b] - 2 * dep[lca(a, b)];
66     }
67 };

```

2.6 Bfs

```
1 vector<vector<int>> adj; // adjacency list
```

```

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

```

2.7 Min Cost Max Flow

```

1 // Min Cost Max Flow (brunomaletta)
2 //
3 // min_cost_flow(s, t, f) computa o par (fluxo, custo)
4 // com max(fluxo) <= f que tenha min(custo)
5 // min_cost_flow(s, t) -> Fluxo maximo de custo
6 // minimo de s pra t
7 // Se for um dag, da pra substituir o SPFA por uma DP
8 // pra nao
9 // pagar O(nm) no comeco
10 // Se nao tiver aresta com custo negativo, nao
11 // precisa do SPFA
12 //
13 // O(nm + f * m log n)
14
15 template<typename T> struct mcmf {
16     struct edge {
17         int to, rev, flow, cap; // para, id da
18         reversa, fluxo, capacidade
19         bool res; // se eh reversa
20         T cost; // custo da unidade de fluxo
21         edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false) {}
22         edge(int to_, int rev_, int flow_, int cap_, T cost_, bool res_)
23             : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_), cost(cost_) {}
24     };
25
26     vector<vector<edge>> g;
27     vector<int> par_idx, par;

```

```

24 T inf;
25 vector<T> dist;
26
27 mcmf(int n) : g(n), par_idx(n), par(n), inf(
numeric_limits<T>::max()/3) {}
28
29 void add(int u, int v, int w, T cost) { // de u
pra v com cap w e custo cost
30     edge a = edge(v, g[v].size(), 0, w, cost,
false);
31     edge b = edge(u, g[u].size(), 0, 0, -cost,
true);
32
33     g[u].push_back(a);
34     g[v].push_back(b);
35 }
36
37 vector<T> spfa(int s) { // nao precisa se nao
tiver custo negativo
38     deque<int> q;
39     vector<bool> is_inside(g.size(), 0);
40     dist = vector<T>(g.size(), inf);
41
42     dist[s] = 0;
43     q.push_back(s);
44     is_inside[s] = true;
45
46     while (!q.empty()) {
47         int v = q.front();
48         q.pop_front();
49         is_inside[v] = false;
50
51         for (int i = 0; i < g[v].size(); i++) {
52             auto [to, rev, flow, cap, res, cost]
= g[v][i];
53             if (flow < cap and dist[v] + cost <
dist[to]) {
54                 dist[to] = dist[v] + cost;
55
56                 if (is_inside[to]) continue;
57                 if (!q.empty() and dist[to] >
dist[q.front()]) q.push_back(to);
58                 else q.push_front(to);
59                 is_inside[to] = true;
60             }
61         }
62     }
63     return dist;
64 }
65 bool dijkstra(int s, int t, vector<T>& pot) {
priority_queue<pair<T, int>, vector<pair<T,
int>>, greater<>> q;
66     dist = vector<T>(g.size(), inf);
67     dist[s] = 0;
68     q.emplace(0, s);
69     while (q.size()) {
70         auto [d, v] = q.top();
71         q.pop();
72         if (dist[v] < d) continue;
73         for (int i = 0; i < g[v].size(); i++) {
74             auto [to, rev, flow, cap, res, cost]
= g[v][i];
75             cost += pot[v] - pot[to];
76             if (flow < cap and dist[v] + cost <
dist[to]) {
77                 dist[to] = dist[v] + cost;
78                 q.emplace(dist[to], to);
79                 par_idx[to] = i, par[to] = v;
80             }
81         }
82     }
83     return dist[t] < inf;
84 }
85

```

```

86
87 pair<int, T> min_cost_flow(int s, int t, int flow
= INF) {
88     vector<T> pot(g.size(), 0);
89     pot = spfa(s); // mudar algoritmo de caminho
minimo aqui
90
91     int f = 0;
92     T ret = 0;
93     while (f < flow and dijkstra(s, t, pot)) {
94         for (int i = 0; i < g.size(); i++)
95             if (dist[i] < inf) pot[i] += dist[i];
96
97         int mn_flow = flow - f, u = t;
98         while (u != s){
99             mn_flow = min(mn_flow,
100                 g[par[u]][par_idx[u]].cap - g[par
[u]][par_idx[u]].flow);
101             u = par[u];
102         }
103
104         ret += pot[t] * mn_flow;
105
106         u = t;
107         while (u != s) {
108             g[par[u]][par_idx[u]].flow += mn_flow
109             g[u][g[par[u]][par_idx[u]].rev].flow
-= mn_flow;
110             u = par[u];
111         }
112
113         f += mn_flow;
114     }
115
116     return make_pair(f, ret);
117 }
118
119 // Opcional: retorna as arestas originais por
onde passa flow = cap
120 vector<pair<int,int>> recover() {
121     vector<pair<int,int>> used;
122     for (int i = 0; i < g.size(); i++) for (edge
e : g[i])
123         if (e.flow == e.cap && !e.res) used.
push_back({i, e.to});
124     return used;
125 }
126 };

```

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<
edge>& edges)
6 {
7     const int INF { 1e9+17 };
8
9     vector<int> dist(N + 1, INF);
10    dist[s] = 0;
11
12    for (int i = 1; i <= N - 1; i++) {
13        for (auto [u, v, w] : edges) {
14            if (dist[u] < INF && dist[v] > dist[u] +
w) {
15                dist[v] = dist[u] + w;
16            }
17        }
18    }
19

```



```

20     for (auto [u, v, w] : edges) {
21         if (dist[u] < INF && dist[v] > dist[u] + w) {
22             return true;
23         }
24     }
25
26     return false;
27 }

```

2.9 3sat

```

1 // We are given a CNF, e.g. phi(x) = (x_1 or ~x_2)
  // and (x_3 or ~x_4 or ~x_5) and ...
2 // SAT finds an assignment x for phi(x) = true.
3 // Davis-Putnam-Logemann-Loveland Algorithm (
  // youknowwho code)
4 // Complexity: O(2^n) in worst case.
5 // This implementation is practical for n <= 1000 or
  // more. lmao.
6
7 #include <bits/stdc++.h>
8 using namespace std;
9
10 const int N = 3e5 + 9;
11
12 // positive literal x in [0,n),
13 // negative literal ~x in [-n,0)
14 // 0 indexed
15 struct SAT_GOD {
16     int n;
17     vector<int> occ, pos, neg;
18     vector<vector<int>> g, lit;
19     SAT_GOD(int n) : n(n), g(2*n), occ(2*n) {}
20     void add_clause(const vector<int> &c) {
21         for(auto u: c) {
22             g[u+n].push_back(lit.size());
23             occ[u+n] += 1;
24         }
25         lit.push_back(c);
26     }
27     //(!u | v | !w) -> (u, 0, v, 1, w, 0)
28     void add(int u, int af, int v = 1e9, int bf = 0,
29             int w = 1e9, int cf = 0) {
30         vector<int> a;
31         if(!af) u = ~u;
32         a.push_back(u);
33         if(v != 1e9) {
34             if(!bf) v = ~v;
35             a.push_back(v);
36         }
37         if(w != 1e9) {
38             if(!cf) w = ~w;
39             a.push_back(w);
40         }
41         add_clause(a);
42     }
43     vector<bool> x;
44     vector<vector<int>> decision_stack;
45     vector<int> unit_stack, pure_stack;
46     void push(int u) {
47         x[u + n] = 1;
48         decision_stack.back().push_back(u);
49         for (auto i: g[u + n]) if (pos[i]++ == 0) {
50             for (auto u: lit[i]) --occ[u+n];
51         }
52         for (auto i: g[~u + n]) {
53             ++neg[i];
54             if (pos[i] == 0) unit_stack.push_back(i);
55         }
56     }
57     void pop() {
58         int u = decision_stack.back().back();
59         decision_stack.back().pop_back();

```

```

59         x[u + n] = 0;
60         for (auto i: g[u + n]) if (--pos[i] == 0) {
61             for (auto u: lit[i]) ++occ[u + n];
62         }
63         for (auto i: g[~u+n]) --neg[i];
64     }
65     bool reduction() {
66         while(!unit_stack.empty() || !pure_stack.empty())
67         {
68             if(!pure_stack.empty()) { // pure literal
69                 elimination
70                 int u = pure_stack.back();
71                 pure_stack.pop_back();
72                 if (occ[u + n] == 1 && occ[~u + n] == 0) push
73                 (u);
74             } else { // unit propagation
75                 int i = unit_stack.back();
76                 unit_stack.pop_back();
77                 if(pos[i] > 0) continue;
78                 if(neg[i] == lit[i].size()) return false;
79                 if(neg[i] + 1 == lit[i].size()) {
80                     int w = n;
81                     for (int u: lit[i]) if (!x[u + n] && !x[~u
82                     + n]) w = u;
83                     if (x[~w + n]) return false;
84                     push(w);
85                 }
86             }
87             return true;
88         }
89     }
90     bool ok() {
91         x.assign(2*n,0);
92         pos = neg = vector<int>(lit.size());
93         decision_stack.assign(1, {});
94         while(1) {
95             if(reduction()) {
96                 int s = 0;
97                 for(int u = 0; u < n; ++u) if(occ[s + n] +
98                 occ[~s + n] < occ[u + n] + occ[~u + n]) s = u;
99                 if(occ[s + n] + occ[~s + n] == 0) return true
100             };
101             decision_stack.push_back({});
102             push(s);
103         } else {
104             int s = decision_stack.back()[0];
105             while(!decision_stack.back().empty()) pop();
106             decision_stack.pop_back();
107             if (decision_stack.empty()) return false;
108             push(~s);
109         }
110     }
111 }
112
113 int32_t main() {
114     int n = 9;
115     SAT_GOD t(n);
116     t.add(0, 0, 1, 1);
117     t.add(1, 0);
118     t.add(1, 0, 3, 1, 5, 1);
119     cout << t.ok() << endl;
120 }

```

2.10 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(to), cap(cap), flow(0) {};
```

```

20
21     ll remaining_cap() {
22         return cap - flow;
23     }
24
25     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         return level[t] != -1;
78     }
79
80     ll dfs(int x, int t, ll flow) {
81         if (x == t) return flow;
82
83         for (int& cid = next[x]; cid < (int)adj[x].
84             size(); cid++) {
85             auto& edge = adj[x][cid];
86             ll cap = edge->remaining_cap();
87
88             if (cap > 0 && level[edge->to] == level[x]
89                 + 1) {
90                 ll sent = dfs(edge->to, t, min(flow,
91                     cap)); // bottle neck
92                 if (sent > 0) {
93                     edge->augment(sent);
94                     return sent;
95                 }
96             }
97         }
98         return 0;
99     }
100
101     ll solve(int s, int t) {
102         ll max_flow = 0;
103
104         while (bfs(s, t)) {
105             fill(next.begin(), next.end(), 0);
106
107             while (ll sent = dfs(s, t, FLOW_INF)) {
108                 max_flow += sent;
109             }
110         }
111
112         return max_flow;
113     }
114
115     // path recover
116     vector<bool> vis;
117     vector<int> curr;
118
119     bool dfs2(int x, int& t) {
120         vis[x] = true;
121         bool arrived = false;
122
123         if (x == t) {
124             curr.push_back(x);
125             return true;
126         }
127
128         for (auto e : adj[x]) {
129             if (e->flow > 0 && !vis[e->to]) { // !e->
130                 is_residual() &&
131                 bool aux = dfs2(e->to, t);
132
133                 if (aux) {
134                     arrived = true;
135                     e->flow--;
136                 }
137             }
138         }
139
140         if (arrived) curr.push_back(x);
141
142         return arrived;
143     }
144
145     vector<vector<int>> get_paths(int s, int t) {
146         vector<vector<int>> ans;
147
148         while (true) {
149             curr.clear();
150             vis.assign(n+1, false);
151         }

```

```

147         if (!dfs2(s, t)) break;
148
149         reverse(curr.begin(), curr.end());
150         ans.push_back(curr);
151     }
152
153     return ans;
154 }
155 };
156

```

3 String

3.1 Trie Xor

```

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

```

```

55     }
56
57     int search(int x) {
58         int curr = 0;
59
60         for (int i = 31; i >= 0; i--) {
61             int b = ((x & (1 << i)) > 0);
62
63             if (trie[curr][b] == 0) return false;
64
65             curr = trie[curr][b];
66         }
67
68         return (finish[curr] > 0);
69     }
70
71     int max_xor(int x) { // maximum xor with x and
  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
  a Ã substring de b
4     for(int i = 0; i < b.size(); i++){
5         int it = i, jt = 0; // b[it], a[jt]
6
7         while(it < b.size() && jt < a.size()){
8             if(b[it] != a[jt])

```

```

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

```

```

51     TripleHash(string s) : hash1(s, MOD1), hash2(s,
52     MOD2), hash3(s, MOD3) {}
53
54     tuple<int, int, int> query(int l, int r) {
55         return { hash1.query(l, r), hash2.query(l, r)
56         , hash3.query(l, r) };
57     }
58
59     tuple<int, int, int> query_inv(int l, int r) {
60         return { hash1.query_inv(l, r), hash2.
61         query_inv(l, r), hash3.query_inv(l, r) };
62     }
63 };
64
65 struct HashK {
66     vector<ll> primes; // more primes = more hashes
67     vector<Hash> hash;
68
69     HashK();
70
71     HashK(string s, vector<ll> primes): primes(primes)
72     {
73         for (auto p : primes) {
74             hash.push_back(Hash(s, p));
75         }
76     }
77
78     vector<int> query(int l, int r) {
79         vector<int> ans;
80
81         for (auto h : hash) {
82             ans.push_back(h.query(l, r));
83         }
84
85         return ans;
86     }
87
88     vector<int> query_inv(int l, int r) {
89         vector<int> ans;
90
91         for (auto h : hash) {
92             ans.push_back(h.query_inv(l, r));
93         }
94
95         return ans;
96     }
97 };

```

4 Math

4.1 Fft Quirino

```

1 // FFT
2 //
3 // boa em memÃria 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     int N = 1;
47     while (N < size(A) + size(B))
48         N <= 1;
49     fa.resize(N);
50     fb.resize(N);
51     fft(fa, false);
52     fft(fb, false);
53     for (int i = 0; i < N; i++)
54         fa[i] *= fb[i];
55     fft(fa, true);
56
57     vector<int> result(N);
58     for (int i = 0; i < N; i++)
59         result[i] = round(fa[i].real());
60     return result;
61 }

```

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 }

```

4.11 Number Sum Product Of Divisors

```

1 // CSES - Divisor Analysis
2 // Print the number, sum and product of the divisors.
3 // Since the input number may be large, it is given
4 // as a prime factorization.
5 // Input:
6 // The first line has an integer n: the number of
7 // parts in the prime factorization.
8 // After this, there are n lines that describe the
9 // factorization. Each line has two numbers x and k
10 // where x is a prime and k is its power.
11 // Output:

```

```

10 // Print three integers modulo 10^9+7: the number,
11 // sum and product of the divisors.
12 // Constraints:
13 // (1 <= n <= 1e5) ; (2 <= x <= 1e6) ; (1 <= k <= 1e9
14 // ) ; each x is a distinct prime
15
16 #include <bits/stdc++.h>
17 typedef long long ll;
18 using namespace std;
19
20 const ll MOD = 1e9 + 7;
21
22 ll expo(ll base, ll pow) {
23     ll ans = 1;
24     while (pow) {
25         if (pow & 1) ans = ans * base % MOD;
26         base = base * base % MOD;
27         pow >>= 1;
28     }
29     return ans;
30 }
31
32 ll p[100001], k[100001];
33
34 int main() {
35     cin.tie(0) -> sync_with_stdio(0);
36     int n;
37     cin >> n;
38     for (int i = 0; i < n; i++) cin >> p[i] >> k[i];
39     ll div_cnt = 1, div_sum = 1, div_prod = 1,
40     div_cnt2 = 1;
41     for (int i = 0; i < n; i++) {
42         div_cnt = div_cnt * (k[i] + 1) % MOD;
43         div_sum = div_sum * (expo(p[i], k[i] + 1) -
44         1) % MOD *
45         expo(p[i] - 1, MOD - 2) % MOD;
46         div_prod = expo(div_prod, k[i] + 1) *
47         expo(expo(p[i], (k[i] * (k[i] + 1)
48         / 2)), div_cnt2) % MOD;
49         div_cnt2 = div_cnt2 * (k[i] + 1) % (MOD - 1);
50     }
51     cout << div_cnt << ' ' << div_sum << ' ' <<
52     div_prod;
53     return 0;
54 }

```

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 //
7 // https://cses.fi/problemset/task/2195/
8 // https://open.kattis.com/problems/convexhull (
9 // counterclockwise)
10
11 typedef long long ftype;
12
13 struct Point {
14     ftype x, y;
15
16     Point() {};
17     Point(ftype x, ftype y) : x(x), y(y) {};
18
19     bool operator<(Point o) {

```

```

20         if (x == o.x) return y < o.y;
21         return x < o.x;
22     }
23
24     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());

```

```

87         // counterclockwise
88     for (int i = (int)lower_hull.size() - 2; i > 0; i
89         --) {
90         upper_hull.push_back(lower_hull[i]);
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 // O(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     return ans;
32 }

```

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) {

```

```

18     if (left > right) return 0;
19
20     int& mem = tb[left][right];
21     if (mem != -1) return mem;
22
23     mem = 1 + dp(left+1, right); // gastar uma
24     operaçãõ arrumando sã o cara atual
25     for (int i = left+1; i <= right; i++) {
26         if (s[left] == s[i]) {
27             mem = min(mem, dp(left+1, i-1) + dp(i,
28             right));
29         }
30     }
31     return mem;
32 }
33
34 int main() {
35     ios::sync_with_stdio(false);
36     cin.tie(NULL);
37
38     cin >> n >> s;
39     memset(tb, -1, sizeof(tb));
40     cout << dp(0, n-1) << '\n';
41
42     return 0;
43 }

```

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
16     under, bool started) {
17     if (pos >= (int)number.size()) {
18         return 1;
19     }

```

```

18     }
19
20     ll& mem = tb[pos][last_digit][under][started];
21     if (mem != -1) return mem;
22     mem = 0;
23
24     int limit = 9;
25     if (!under) limit = number[pos] - '0';
26
27     for (int digit = 0; digit <= limit; digit++) {
28         if (started && digit == last_digit) continue;
29
30         bool is_under = under || (digit < limit);
31         bool is_started = started || (digit != 0);
32
33         mem += dp(number, pos+1, digit, is_under,
34         is_started);
35     }
36     return mem;
37 }
38
39 ll solve(ll ubound) {
40     memset(tb, -1, sizeof(tb));
41     string number = to_string(ubound);
42     return dp(number, 0, 10, 0, 0);
43 }
44
45 int main() {
46     ios::sync_with_stdio(false);
47     cin.tie(NULL);
48
49     ll a, b; cin >> a >> b;
50     cout << solve(b) - solve(a-1) << '\n';
51
52     return 0;
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
6          ) - dp.begin();
7
8          if (pos == (int)dp.size()) {
9              dp.push_back(e);
10             } else {
11                 dp[pos] = e;
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
8  // resposta
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 Flags

```

1 // g++ -std=c++17 -Wall -Wshadow -fsanitize=address -
  02 -D -o cod a.cpp

```

7.6 Input By File

```

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

```

7.7 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 }

```

7.8 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
8 // + 1
9 //
10 // f(4) = false
11 // f(5) = false
12 // f(6) = true
13 // f(7) = true
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 }

```

7.9 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 }

```

7.10 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
8 // - 1
9 //
10 // f(1) = true
11 // f(2) = true
12 // f(3) = true
13 // f(4) = true
14 // f(5) = true
15 // f(6) = false
16 // f(7) = false
17 // f(8) = false
18 //

```

```

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;
26
27         if (f(mid)) {
28             lo = mid;
29         } else {
30             hi = mid - 1;
31         }
32     }
33     return lo;
34 }

```

7.11 Overflow

```

1 // Signatures of some built-in functions to perform
2 // arithmetic operations with overflow check
3 // Source: https://gcc.gnu.org/onlinedocs/gcc/Integer-Overflow-Builtins.html
4 //
5 // you can also check overflow by performing the
6 // operation with double
7 // and checking if the result it's greater than the
8 // maximum value supported by the variable
9
10 bool __builtin_add_overflow (type1 a, type2 b, type3
11                             *res)
12 bool __builtin_sadd_overflow (int a, int b, int *res)
13 bool __builtin_saddl_overflow (long int a, long int b
14                               , long int *res)
15 bool __builtin_saddll_overflow (long long int a, long
16                                long int b, long long int *res)
17 bool __builtin_uadd_overflow (unsigned int a,
18                               unsigned int b, unsigned int *res)
19 bool __builtin_uaddl_overflow (unsigned long int a,
20                               unsigned long int b, unsigned long int *res)
21 bool __builtin_uaddll_overflow (unsigned long long
22                                int a, unsigned long long
23                                int b, unsigned long
24                                long int *res)
25
26 bool __builtin_sub_overflow (type1 a, type2 b, type3
27                             *res)
28 bool __builtin_ssub_overflow (int a, int b, int *res)
29 bool __builtin_ssubl_overflow (long int a, long int b
30                               , long int *res)
31 bool __builtin_ssubll_overflow (long long int a, long
32                                long int b, long long int *res)
33 bool __builtin_usub_overflow (unsigned int a,
34                               unsigned int b, unsigned int *res)
35 bool __builtin_usubl_overflow (unsigned long int a,
36                               unsigned long int b, unsigned long int *res)
37 bool __builtin_usubll_overflow (unsigned long long
38                                int a, unsigned long long
39                                int b, unsigned long
40                                long int *res)
41
42 bool __builtin_mul_overflow (type1 a, type2 b, type3
43                             *res)
44 bool __builtin_smul_overflow (int a, int b, int *res)

```

```

25 bool __builtin_smull_overflow (long int a, long int b
26                               , long int *res)
27 bool __builtin_smulll_overflow (long long int a, long
28                                long int b, long long int *res)
29 bool __builtin_umul_overflow (unsigned int a,
30                               unsigned int b, unsigned int *res)
31 bool __builtin_umull_overflow (unsigned long int a,
32                               unsigned long int b, unsigned long int *res)
33 bool __builtin_umulll_overflow (unsigned long long
34                                int a, unsigned long long
35                                int b, unsigned long
36                                long int *res)
37
38 bool __builtin_add_overflow_p (type1 a, type2 b,
39                                type3 c)
40 bool __builtin_sub_overflow_p (type1 a, type2 b,
41                                type3 c)
42 bool __builtin_mul_overflow_p (type1 a, type2 b,
43                                type3 c)

```

7.12 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 }

```

7.13 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()));

```

7.14 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
7 int val = dist(rng);

```

8 Primitives