Competitive programming Notebook

Meia noite eu te conto

Contents

1 String **2**
- 1.1 Is Substring . . . 2
- 1.2 Triexor . . . 2

2 Graph **2**
- 2.1 Lca . . . 2
- 2.2 Ford Fulkerson . . . 3
- 2.3 Dinic . . . 3
- 2.4 Floyd Warshall . . . 4
- 2.5 Has Negative Cycle . . . 5
- 2.6 Is Prime . . . 5
- 2.7 Bfs . . . 5
- 2.8 Dijkstra . . . 5
- 2.9 Kruskal . . . 5
- 2.10 Dfs . . . 6

3 Math **6**
- 3.1 Divisors . . . 6
- 3.2 Fexp . . . 6
- 3.3 Factorization . . . 6
- 3.4 Ceil . . . 6
- 3.5 Log Any Base . . . 6
- 3.6 2sat . . . 7
- 3.7 Sieve . . . 8
- 3.8 Generate Primes . . . 8

4 DP **8**
- 4.1 Lis Binary Search . . . 8
- 4.2 Digit Dp . . . 8
- 4.3 Lcs . . . 8
- 4.4 Digit Dp 2 . . . 9
- 4.5 Lis Segtree . . . 9
- 4.6 Knapsack . . . 10
- 4.7 Range Dp . . . 10
- 4.8 Edit Distance . . . 10

5 Geometry **11**
- 5.1 Convex Hull . . . 11

6 General **11**
- 6.1 Base Converter . . . 11
- 6.2 First True . . . 12
- 6.3 Random . . . 12
- 6.4 Bitwise . . . 12
- 6.5 Min Priority Queue . . . 12
- 6.6 Input By File . . . 12
- 6.7 Read . . . 12
- 6.8 Compilation Flags . . . 12
- 6.9 Xor 1 To N . . . 12
- 6.10 Next Permutation . . . 12
- 6.11 Template Full . . . 12
- 6.12 Goto . . . 13
- 6.13 Split . . . 13
- 6.14 Last True . . . 13
- 6.15 Template . . . 13
- 6.16 Ordered Set . . . 13
- 6.17 Get Subset Sums . . . 14

7 DS **14**
- 7.1 Segtree2d . . . 14
- 7.2 Minimum And Amount . . . 15
- 7.3 Segment With Maximum Sum . . . 16
- 7.4 Persistent . . . 17
- 7.5 Range Query Point Update . . . 18
- 7.6 Lazy Assignment To Segment . . . 19
- 7.7 Dynamic Implicit Sparse . . . 20
- 7.8 Lazy Addition To Segment . . . 20
- 7.9 Lazy Dynamic Implicit Sparse . . . 21
- 7.10 Big K . . . 22
- 7.11 Ordered Set . . . 23
- 7.12 Dsu . . . 23

1 String

1.1 Is Substring

```cpp
// equivalente ao in do python

bool is_substring(string a, string b){ // verifica se a Ã© substring de b
    for(int i = 0; i < b.size(); i++){
        int it = i, jt = 0; // b[it], a[jt]

        while(it < b.size() && jt < a.size()){
            if(b[it] != a[jt])
                break;

            it++;
            jt++;

            if(jt == a.size())
                return true;
        }
    }

    return false;
}
```

1.2 Triexor

```cpp
// TrieXOR
//
// adiciona, remove e verifica se existe strings binarias
// max_xor(x) = maximiza o xor de x com algum valor da trie
//
// raiz = 0
//
// https://codeforces.com/problemset/problem/706/D
//
// O(|s|) adicionar, remover e buscar

struct TrieXOR {
    int n, alph_sz, nxt;
    vector<vector<int>> trie;
    vector<int> finish, paths;

    TrieXOR() {}

    TrieXOR(int n, int alph_sz = 2) : n(n), alph_sz(alph_sz) {
        nxt = 1;
        trie.assign(n, vector<int>(alph_sz));
        finish.assign(n * alph_sz, 0);
        paths.assign(n * alph_sz, 0);
    }

    void add(int x) {
        int curr = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);

            if (trie[curr][b] == 0)
                trie[curr][b] = nxt++;

            paths[curr]++;
            curr = trie[curr][b];
        }

        paths[curr]++;
        finish[curr]++;
    }

    void rem(int x) {
        int curr = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);

            paths[curr]--;
            curr = trie[curr][b];
        }

        paths[curr]--;
        finish[curr]--;
    }

    int search(int x) {
        int curr = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);

            if (trie[curr][b] == 0) return false;

            curr = trie[curr][b];
        }

        return (finish[curr] > 0);
    }

    int max_xor(int x) { // maximum xor with x and any number of trie
        int curr = 0, ans = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);
            int want = b^1;

            if (trie[curr][want] == 0 || paths[trie[curr][want]] == 0) want ^= 1;
            if (trie[curr][want] == 0 || paths[trie[curr][want]] == 0) break;
            if (want != b) ans |= (1 << i);

            curr = trie[curr][want];
        }

        return ans;
    }
};
```

2 Graph

2.1 Lca

```cpp
// LCA
//
// lowest common ancestor between two nodes
//
// edit_distance(n, adj, root)
//
// https://cses.fi/problemset/task/1688
//
// O(log N)

struct LCA {
  const int MAXE = 31;
  vector<vector<int>> up;
  vector<int> dep;

  LCA(int n, vector<vector<int>>& adj, int root = 1) {
    up.assign(n+1, vector<int>(MAXE, -1));
```

```
    dep.assign(n+1, 0);

    dep[root] = 1;
    dfs(root, -1, adj);

    for (int j = 1; j < MAXE; j++) {
      for (int i = 1; i <= n; i++) {
        if (up[i][j-1] != -1)
          up[i][j] = up[ up[i][j-1] ][j-1];
      }
    }
  }

  void dfs(int x, int p, vector<vector<int>>& adj) {
    up[x][0] = p;
    for (auto e : adj[x]) {
      if (e != p) {
        dep[e] = dep[x] + 1;
        dfs(e, x, adj);
      }
    }
  }

  int jump(int x, int k) { // jump from node x k
    times
    for (int i = 0; i < MAXE; i++) {
      if (k&(1 << i) && x != -1) x = up[x][i];
    }
    return x;
  }

  int lca(int a, int b) {
    if (dep[a] > dep[b]) swap(a, b);
    b = jump(b, dep[b] - dep[a]);

    if (a == b) return a;

    for (int i = MAXE-1; i >= 0; i--) {
      if (up[a][i] != up[b][i]) {
        a = up[a][i];
        b = up[b][i];
      }
    }

    return up[a][0];
  }

  int dist(int a, int b) {
    return dep[a] + dep[b] - 2 * dep[lca(a, b)];
  }
};
```

2.2 Ford Fulkerson

```
// Ford-Fulkerson
//
// max-flow / min-cut
//
// MAX nÃşs
//
// https://cses.fi/problemset/task/1694/
//
// O(m * max_flow)

using ll = long long;
const int MAX = 510;

struct Flow {
    int n;
    ll adj[MAX][MAX];
    bool used[MAX];

    Flow(int n) : n(n) {};

    void add_edge(int u, int v, ll c) {
        adj[u][v] += c;
        adj[v][u] = 0; // cuidado com isso
    }

    ll dfs(int x, int t, ll amount) {
        used[x] = true;

        if (x == t) return amount;

        for (int i = 1; i <= n; i++) {
            if (adj[x][i] > 0 && !used[i]) {
                ll sent = dfs(i, t, min(amount, adj[x
    ][i]));

                if (sent > 0) {
                    adj[x][i] -= sent;
                    adj[i][x] += sent;

                    return sent;
                }
            }
        }

        return 0;
    }

    ll max_flow(int s, int t) { // source and sink
        ll total = 0;
        ll sent = -1;

        while (sent != 0) {
            memset(used, 0, sizeof(used));
            sent = dfs(s, t, INT_MAX);
            total += sent;
        }

        return total;
    }
};
```

2.3 Dinic

```
// Dinic / Dinitz
//
// max-flow / min-cut
//
// https://cses.fi/problemset/task/1694/
//
// O(E * V^2)

using ll = long long;
const ll FLOW_INF = 1e18 + 7;

struct Edge {
    int from, to;
    ll cap, flow;
    Edge* residual; // a inversa da minha aresta

    Edge() {};

    Edge(int from, int to, ll cap) : from(from), to(
    to), cap(cap), flow(0) {};

    ll remaining_cap() {
        return cap - flow;
    }

    void augment(ll bottle_neck) {
        flow += bottle_neck;
        residual->flow -= bottle_neck;
    }
```

```
    bool is_residual() {
        return cap == 0;
    }
};

struct Dinic {
    int n;
    vector<vector<Edge*>> adj;
    vector<int> level, next;

    Dinic(int n): n(n) {
        adj.assign(n+1, vector<Edge*>());
        level.assign(n+1, -1);
        next.assign(n+1, 0);
    }

    void add_edge(int from, int to, ll cap) {
        auto e1 = new Edge(from, to, cap);
        auto e2 = new Edge(to, from, 0);

        e1->residual = e2;
        e2->residual = e1;

        adj[from].push_back(e1);
        adj[to].push_back(e2);
    }

    bool bfs(int s, int t) {
        fill(level.begin(), level.end(), -1);
        queue<int> q;

        q.push(s);
        level[s] = 1;

        while (q.size()) {
            int curr = q.front();
            q.pop();

            for (auto edge : adj[curr]) {
                if (edge->remaining_cap() > 0 && level[edge->to] == -1) {
                    level[edge->to] = level[curr] + 1;
                    q.push(edge->to);
                }
            }
        }

        return level[t] != -1;
    }

    ll dfs(int x, int t, ll flow) {
        if (x == t) return flow;

        for (int& cid = next[x]; cid < (int)adj[x].size(); cid++) {
            auto& edge = adj[x][cid];
            ll cap = edge->remaining_cap();

            if (cap > 0 && level[edge->to] == level[x] + 1) {
                ll sent = dfs(edge->to, t, min(flow, cap)); // bottle neck
                if (sent > 0) {
                    edge->augment(sent);
                    return sent;
                }
            }
        }

        return 0;
    }

    ll solve(int s, int t) {
        ll max_flow = 0;

        while (bfs(s, t)) {
            fill(next.begin(), next.end(), 0);

            while (ll sent = dfs(s, t, FLOW_INF)) {
                max_flow += sent;
            }
        }

        return max_flow;
    }

    // path recover
    vector<bool> vis;
    vector<int> curr;

    bool dfs2(int x, int& t) {
        vis[x] = true;
        bool arrived = false;

        if (x == t) {
            curr.push_back(x);
            return true;
        }

        for (auto e : adj[x]) {
            if (e->flow > 0 && !vis[e->to]) { // !e->is_residual() &&
                bool aux = dfs2(e->to, t);

                if (aux) {
                    arrived = true;
                    e->flow--;
                }
            }
        }

        if (arrived) curr.push_back(x);

        return arrived;
    }

    vector<vector<int>> get_paths(int s, int t) {
        vector<vector<int>> ans;

        while (true) {
            curr.clear();
            vis.assign(n+1, false);

            if (!dfs2(s, t)) break;

            reverse(curr.begin(), curr.end());
            ans.push_back(curr);
        }

        return ans;
    }
};
```

2.4 Floyd Warshall

```
const long long LLINF = 0x3f3f3f3f3f3f3f3fLL;

for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        adj[i][j] = 0;
    }
}

long long dist[MAX][MAX];
```

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        if (i == j)
            dist[i][j] = 0;
        else if (adj[i][j])
            dist[i][j] = adj[i][j];
        else
            dist[i][j] = LLINF;
    }
}

for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
        }
    }
}
```

2.5 Has Negative Cycle

```
// Edson

using edge = tuple<int, int, int>;

bool has_negative_cycle(int s, int N, const vector<edge>& edges)
{
    const int INF { 1e9+17 };

    vector<int> dist(N + 1, INF);
    dist[s] = 0;

    for (int i = 1; i <= N - 1; i++) {
        for (auto [u, v, w] : edges) {
            if (dist[u] < INF && dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
            }
        }
    }

    for (auto [u, v, w] : edges) {
        if (dist[u] < INF && dist[v] > dist[u] + w) {
            return true;
        }
    }

    return false;
}
```

2.6 Is Prime

```
bool is_prime(ll n) {
    if (n <= 1) return false;
    if (n == 2) return true;

    for (ll i = 2; i*i <= n; i++) {
        if (n % i == 0)
            return false;
    }

    return true;
}
```

2.7 Bfs

```
vector<vector<int>> adj; // adjacency list representation
int n; // number of nodes
int s; // source vertex

queue<int> q;
vector<bool> used(n + 1);
vector<int> d(n + 1), p(n + 1);

q.push(s);
used[s] = true;
p[s] = -1;
while (!q.empty()) {
    int v = q.front();
    q.pop();
    for (int u : adj[v]) {
        if (!used[u]) {
            used[u] = true;
            q.push(u);
            d[u] = d[v] + 1;
            p[u] = v;
        }
    }
}

// restore path
if (!used[u]) {
    cout << "No path!";
} else {
    vector<int> path;

    for (int v = u; v != -1; v = p[v])
        path.push_back(v);

    reverse(path.begin(), path.end());

    cout << "Path: ";
    for (int v : path)
        cout << v << " ";
}
```

2.8 Dijkstra

```
const int INF = 1e9+17;
vector<vector<pair<int, int>>> adj; // {neighbor, weight}

void dijkstra(int s, vector<int> & d, vector<int> & p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);

    d[s] = 0;
    set<pair<int, int>> q;
    q.insert({0, s});
    while (!q.empty()) {
        int v = q.begin()->second;
        q.erase(q.begin());

        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                q.erase({d[to], to});
                d[to] = d[v] + len;
                p[to] = v;
                q.insert({d[to], to});
            }
        }
    }
}
```

2.9 Kruskal

```cpp
// need: DSU

struct Edge {
    int u, v;
    ll weight;

    Edge() {}

    Edge(int u, int v, ll weight) : u(u), v(v), weight(weight) {}

    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

vector<Edge> kruskal(vector<Edge> edges, int n) {
    vector<Edge> result;
    ll cost = 0;

    sort(edges.begin(), edges.end());
    DSU dsu(n);

    for (auto e : edges) {
        if (!dsu.same(e.u, e.v)) {
            cost += e.weight;
            result.push_back(e);
            dsu.unite(e.u, e.v);
        }
    }

    return result;
}
```

2.10 Dfs

```cpp
// DFS
//
// Percorre todos os vertices
// priorizando profundidade
//
// O(n+m)

vector<vector<int>> g;
vector<bool> vis;

void dfs(int s){
    if(vis[s]) return;
    vis[s] = true;
    for(auto v : g[s]){
        dfs(v);
    }
}
```

3 Math

3.1 Divisors

```cpp
vector<ll> divisors(ll n) {
    vector<ll> ans;

    for (ll i = 1; i*i <= n; i++) {
        if (n%i == 0) {
            ll value = n/i;

            ans.push_back(i);
            if (value != i) {
                ans.push_back(value);
            }
        }
    }

    return ans;
}
```

3.2 Fexp

```cpp
using ll = long long;

ll fexp(ll base, ll exp, ll m) {
    ll ans = 1;
    base %= m;

    while (exp > 0) {
        if (exp % 2 == 1) {
            ans = (ans * base) % m;
        }

        base = (base * base) % m;
        exp /= 2;
    }

    return ans;
}
```

3.3 Factorization

```cpp
// nson

using ll = long long;

vector<pair<ll, int>> factorization(ll n) {
    vector<pair<ll, int>> ans;

    for (ll p = 2; p*p <= n; p++) {
        if (n%p == 0) {
            int expoente = 0;

            while (n%p == 0) {
                n /= p;
                expoente++;
            }

            ans.push_back({p, expoente});
        }
    }

    if (n > 1) {
        ans.push_back({n, 1});
    }

    return ans;
}
```

3.4 Ceil

```cpp
using ll = long long;

// avoid overflow
ll division_ceil(ll a, ll b) {
    return 1 + ((a - 1) / b); // if a != 0
}

int intceil(int a, int b) {
    return (a+b-1)/b;
}
```

3.5 Log Any Base

```cpp
int intlog(double base, double x) {
    return (int)(log(x) / log(base));
}
```

3.6 2sat

```cpp
// 2SAT
//
// verifica se existe e encontra soluÃğÃčo
// para fÃşrmulas booleanas da forma
// (a or b) and (!a or c) and (...)
//
// indexado em 0
// n(a) = 2*x e n(~a) = 2*x+1
// a = 2 ; n(a) = 4 ; n(~a) = 5 ; n(a)^1 = 5 ; n(~a)^1 = 4
//
// https://cses.fi/problemset/task/1684/
// https://codeforces.com/gym/104120/problem/E
// (add_eq, add_true, add_false e at_most_one nÃčo foram testadas)
//
// O(n + m)

struct sat {
    int n, tot;
    vector<vector<int>> adj, adjt; // grafo original, grafo transposto
    vector<int> vis, comp, ans;
    stack<int> topo; // ordem topolÃşgica

    sat() {}
    sat(int n_) : n(n_), tot(n), adj(2*n), adjt(2*n) {}

    void dfs(int x) {
        vis[x] = true;

        for (auto e : adj[x]) {
            if (!vis[e]) dfs(e);
        }

        topo.push(x);
    }

    void dfst(int x, int& id) {
        vis[x] = true;
        comp[x] = id;

        for (auto e : adjt[x]) {
            if (!vis[e]) dfst(e, id);
        }
    }

    void add_impl(int a, int b) { // a -> b = (!a or b)
        a = (a >= 0 ? 2*a : -2*a-1);
        b = (b >= 0 ? 2*b : -2*b-1);

        adj[a].push_back(b);
        adj[b^1].push_back(a^1);

        adjt[b].push_back(a);
        adjt[a^1].push_back(b^1);
    }

    void add_or(int a, int b) { // a or b
        add_impl(~a, b);
    }

    void add_nor(int a, int b) { // a nor b = !(a or b)
        add_or(~a, b), add_or(a, ~b), add_or(~a, ~b);
    }

    void add_and(int a, int b) { // a and b
        add_or(a, b), add_or(~a, b), add_or(a, ~b);
    }

    void add_nand(int a, int b) { // a nand b = !(a and b)
        add_or(~a, ~b);
    }

    void add_xor(int a, int b) { // a xor b = (a != b)
        add_or(a, b), add_or(~a, ~b);
    }

    void add_xnor(int a, int b) { // a xnor b = !(a xor b) = (a = b)
        add_xor(~a, b);
    }

    void add_true(int a) { // a = T
        add_or(a, ~a);
    }

    void add_false(int a) { // a = F
        add_and(a, ~a);
    }

    // magia - brunomaletta
    void add_true_old(int a) { // a = T (n sei se funciona)
        add_impl(~a, a);
    }

    void at_most_one(vector<int> v) { // no max um verdadeiro
        adj.resize(2*(tot+v.size()));
        for (int i = 0; i < v.size(); i++) {
            add_impl(tot+i, ~v[i]);
            if (i) {
                add_impl(tot+i, tot+i-1);
                add_impl(v[i], tot+i-1);
            }
        }
        tot += v.size();
    }

    pair<bool, vector<int>> solve() {
        ans.assign(n, -1);
        comp.assign(2*tot, -1);
        vis.assign(2*tot, 0);
        int id = 1;

        for (int i = 0; i < 2*tot; i++) if (!vis[i]) dfs(i);

        vis.assign(2*tot, 0);
        while (topo.size()) {
            auto x = topo.top();
            topo.pop();

            if (!vis[x]) {
                dfst(x, id);
                id++;
            }
        }

        for (int i = 0; i < tot; i++) {
            if (comp[2*i] == comp[2*i+1]) return {false, {}};
            ans[i] = (comp[2*i] > comp[2*i+1]);
        }

        return {true, ans};
    }
};
```

3.7 Sieve

```cpp
// nao "otimizado"

vector<bool> sieve(int lim=1e5+17) {
    vector<bool> isprime(lim+1, true);

    isprime[0] = isprime[1] = false;

    for (int i = 2; i*i < lim; i++) {
        if (isprime[i]) {
            for (int j = i+i; j < lim; j += i) {
                isprime[j] = false;
            }
        }
    }

    return isprime;
}
```

3.8 Generate Primes

```cpp
// crivo nao otimizado

vector<int> generate_primes(int lim=1e5+17) {
    vector<int> primes;
    vector<bool> isprime(lim+1, true);

    isprime[0] = isprime[1] = false;

    for (int i = 2; i*i < lim; i++) {
        if (isprime[i]) {
            primes.push_back(i);

            for (int j = i+i; j < lim; j += i) {
                isprime[j] = false;
            }
        }
    }

    return primes;
}
```

4 DP

4.1 Lis Binary Search

```cpp
int lis(vector<int> arr) {
    vector<int> dp;

    for (auto e : arr) {
        int pos = lower_bound(dp.begin(), dp.end(), e) - dp.begin();

        if (pos == (int)dp.size()) {
            dp.push_back(e);
        } else {
            dp[pos] = e;
        }
    }

    return (int)dp.size();
}
```

4.2 Digit Dp

```cpp
// Digit DP 1: https://atcoder.jp/contests/dp/tasks/dp_s
//
// find the number of integers between 1 and K (inclusive)
// where the sum of digits in base ten is a multiple of D

#include <bits/stdc++.h>

using namespace std;

const int MOD = 1e9+7;

string k;
int d;

int tb[10010][110][2];

int dp(int pos, int sum, bool under) {
    if (pos >= k.size()) return sum == 0;

    int& mem = tb[pos][sum][under];
    if (mem != -1) return mem;
    mem = 0;

    int limit = 9;
    if (!under) limit = k[pos] - '0';

    for (int digit = 0; digit <= limit; digit++) {
        mem += dp(pos+1, (sum + digit) % d, under | (digit < limit));
        mem %= MOD;
    }

    return mem;
}

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

    cin >> k >> d;

    memset(tb, -1, sizeof(tb));

    cout << (dp(0, 0, false) - 1 + MOD) % MOD << '\n';

    return 0;
}
```

4.3 Lcs

```cpp
// LCS (Longest Common Subsequence)
//
// maior subsequencia comum entre duas strings
//
// tamanho da matriz da dp eh |a| x |b|
// lcs(a, b) = string da melhor resposta
// dp[a.size()][b.size()] = tamanho da melhor resposta
//
// https://atcoder.jp/contests/dp/tasks/dp_f
//
// O(n^2)

string lcs(string a, string b) {
    int n = a.size();
    int m = b.size();

    int dp[n+1][m+1];
    pair<int, int> p[n+1][m+1];

    memset(dp, 0, sizeof(dp));
    memset(p, -1, sizeof(p));

    for (int i = 1; i <= n; i++) {
```

```
        for (int j = 1; j <= m; j++) {
            if (a[i-1] == b[j-1]) {
                dp[i][j] = dp[i-1][j-1] + 1;
                p[i][j] = {i-1, j-1};
            } else {
                if (dp[i-1][j] > dp[i][j-1]) {
                    dp[i][j] = dp[i-1][j];
                    p[i][j] = {i-1, j};
                } else {
                    dp[i][j] = dp[i][j-1];
                    p[i][j] = {i, j-1};
                }
            }
        }
    }

    // recuperar resposta

    string ans = "";
    pair<int, int> curr = {n, m};

    while (curr.first != 0 && curr.second != 0) {
        auto [i, j] = curr;

        if (a[i-1] == b[j-1]) {
            ans += a[i-1];
        }

        curr = p[i][j];
    }

    reverse(ans.begin(), ans.end());

    return ans;
}
```

4.4 Digit Dp 2

```
// Digit DP 2: https://cses.fi/problemset/task/2220
//
// Number of integers between a and b
// where no two adjacents digits are the same

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

const int MAX = 20; // 10^18

ll tb[MAX][MAX][2][2];

ll dp(string& number, int pos, int last_digit, bool under, bool started) {
    if (pos >= (int)number.size()) {
        return 1;
    }

    ll& mem = tb[pos][last_digit][under][started];
    if (mem != -1) return mem;
    mem = 0;

    int limit = 9;
    if (!under) limit = number[pos] - '0';

    for (int digit = 0; digit <= limit; digit++) {
        if (started && digit == last_digit) continue;

        bool is_under = under || (digit < limit);
        bool is_started = started || (digit != 0);

        mem += dp(number, pos+1, digit, is_under, is_started);
    }

    return mem;
}

ll solve(ll ubound) {
    memset(tb, -1, sizeof(tb));
    string number = to_string(ubound);
    return dp(number, 0, 10, 0, 0);
}

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

    ll a, b; cin >> a >> b;
    cout << solve(b) - solve(a-1) << '\n';

    return 0;
}
```

4.5 Lis Segtree

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

using namespace std;

const int MAX = 2e5+17;

struct segTree {
    int size;
    vector<int> tree;

    void init(int n) {
        size = 1;
        while (size < n) size *= 2;
        tree.assign(2 * size, 0LL);
    }

    int merge(int a, int b) {
        return max(a, b);
    }

    void build(vector<int> &arr, int x, int lx, int rx) {
        if (rx - lx == 1) {
            if (lx < (int)arr.size())
                tree[x] = arr[lx];

            return;
        }

        int m = (lx + rx) / 2;
        build(arr, 2 * x + 1, lx, m);
        build(arr, 2 * x + 2, m, rx);

        tree[x] = merge(tree[2 * x + 1], tree[2 * x + 2]);
    }

    void build(vector<int> &arr) {
        build(arr, 0, 0, size);
    }

    void update(int i, int v, int x, int lx, int rx) {
        if (rx - lx == 1) {
            tree[x] = v;
            return;
        }

        int m = (lx + rx) / 2;
        if (i < m) {
```

```
            update(i, v, 2 * x + 1, lx, m);
        } else {
            update(i, v, 2 * x + 2, m, rx);
        }

        tree[x] = merge(tree[2 * x + 1], tree[2 * x + 2]);
    }

    void update(int i, int v) {
        update(i, v, 0, 0, size);
    }

    int query(int l, int r, int x, int lx, int rx) {
        if (lx >= r || l >= rx) return 0;
        if (lx >= l && rx <= r) return tree[x];

        int m = (lx + rx) / 2;
        int s1 = query(l, r, 2 * x + 1, lx, m);
        int s2 = query(l, r, 2 * x + 2, m, rx);

        return merge(s1, s2);
    }

    int query(int l, int r) {
        return query(l, r, 0, 0, size);
    }
};


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

    int n, arr[MAX], aux[MAX]; cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];

        aux[i] = arr[i];
    }

    sort(aux, aux+n);

    segTree st;
    st.init(n);

    int ans = 0;
    for (int i = 0; i < n; i++) {
        int it = lower_bound(aux, aux+n, arr[i]) - aux;
        int lis = st.query(0, it) + 1;

        st.update(it, lis);

        ans = max(ans, lis);
    }

    cout << ans << '\n';

    return 0;
}
```

4.6 Knapsack

4.7 Range Dp

```
// Range DP 1: https://codeforces.com/problemset/problem/1132/F
//
// You may apply some operations to this string
// in one operation you can delete some contiguous substring of this string
// if all letters in the substring you delete are equal
// calculate the minimum number of operations to delete the whole string s

#include <bits/stdc++.h>

using namespace std;

const int MAX = 510;

int n, tb[MAX][MAX];
string s;

int dp(int left, int right) {
    if (left > right) return 0;

    int& mem = tb[left][right];
    if (mem != -1) return mem;

    mem = 1 + dp(left+1, right); // gastar uma operaÃğÃčo arrumando sÃş o cara atual
    for (int i = left+1; i <= right; i++) {
        if (s[left] == s[i]) {
            mem = min(mem, dp(left+1, i-1) + dp(i, right));
        }
    }

    return mem;
}

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

    cin >> n >> s;
    memset(tb, -1, sizeof(tb));
    cout << dp(0, n-1) << '\n';

    return 0;
}
```

4.8 Edit Distance

```
// Edit Distance / Levenshtein Distance
//
// numero minimo de operacoes
// para transformar
// uma string em outra
//
// tamanho da matriz da dp eh |a| x |b|
// edit_distance(a.size(), b.size(), a, b)
//
// https://cses.fi/problemset/task/1639
//
// O(n^2)

int tb[MAX][MAX];

int edit_distance(int i, int j, string &a, string &b) {
    if (i == 0) return j;
    if (j == 0) return i;

    int &ans = tb[i][j];

    if (ans != -1) return ans;

    ans = min({
        edit_distance(i-1, j, a, b) + 1,
        edit_distance(i, j-1, a, b) + 1,
```

```
        edit_distance(i-1, j-1, a, b) + (a[i-1] != b[j-1])
    });

    return ans;
}
```

5 Geometry

5.1 Convex Hull

```
// Convex Hull - Monotone Chain
//
// Convex Hull is the subset of points that forms the smallest convex polygon
// which encloses all points in the set.
//
// https://cses.fi/problemset/task/2195/
// https://open.kattis.com/problems/convexhull (counterclockwise)
//
// O(n log(n))

typedef long long ftype;

struct Point {
    ftype x, y;

    Point() {};
    Point(ftype x, ftype y) : x(x), y(y) {};

    bool operator<(Point o) {
        if (x == o.x) return y < o.y;
        return x < o.x;
    }

    bool operator==(Point o) {
        return x == o.x && y == o.y;
    }
};

ftype cross(Point a, Point b, Point c) {
    // v: a -> c
    // w: a -> b

    // v: c.x - a.x, c.y - a.y
    // w: b.x - a.x, b.y - a.y

    return (c.x - a.x) * (b.y - a.y) - (c.y - a.y) * (b.x - a.x);
}

ftype dir(Point a, Point b, Point c) {
    // 0 -> colineares
    // -1 -> esquerda
    // 1 -> direita

    ftype cp = cross(a, b, c);

    if (cp == 0) return 0;
    else if (cp < 0) return -1;
    else return 1;
}

vector<Point> convex_hull(vector<Point> points) {
    sort(points.begin(), points.end());
    points.erase( unique(points.begin(), points.end()), points.end()); // somente pontos distintos
    int n = points.size();

    if (n == 1) return { points[0] };

    vector<Point> upper_hull = {points[0], points[1]};
    for (int i = 2; i < n; i++) {
        upper_hull.push_back(points[i]);

        int sz = upper_hull.size();

        while (sz >= 3 && dir(upper_hull[sz-3], upper_hull[sz-2], upper_hull[sz-1]) == -1) {
            upper_hull.pop_back();
            upper_hull.pop_back();
            upper_hull.push_back(points[i]);
            sz--;
        }
    }

    vector<Point> lower_hull = {points[n-1], points[n-2]};
    for (int i = n-3; i >= 0; i--) {
        lower_hull.push_back(points[i]);

        int sz = lower_hull.size();

        while (sz >= 3 && dir(lower_hull[sz-3], lower_hull[sz-2], lower_hull[sz-1]) == -1) {
            lower_hull.pop_back();
            lower_hull.pop_back();
            lower_hull.push_back(points[i]);
            sz--;
        }
    }

    // reverse(lower_hull.begin(), lower_hull.end()); // counterclockwise

    for (int i = (int)lower_hull.size() - 2; i > 0; i--) {
        upper_hull.push_back(lower_hull[i]);
    }

    return upper_hull;
}
```

6 General

6.1 Base Converter

```
const string digits = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

ll tobase10(string number, int base) {
    map<char, int> val;
    for (int i = 0; i < digits.size(); i++) {
        val[digits[i]] = i;
    }

    ll ans = 0, pot = 1;

    for (int i = number.size() - 1; i >= 0; i--) {
        ans += val[number[i]] * pot;
        pot *= base;
    }

    return ans;
}

string frombase10(ll number, int base) {
    if (number == 0) return "0";

    string ans = "";

    while (number > 0) {
```

```
        ans += digits[number % base];
        number /= base;
    }

    reverse(ans.begin(), ans.end());

    return ans;
}

// verifica se um nÃžmero estÃą na base especificada
bool verify_base(string num, int base) {
    map<char, int> val;
    for (int i = 0; i < digits.size(); i++) {
        val[digits[i]] = i;
    }

    for (auto digit : num) {
        if (val[digit] >= base) {
            return false;
        }
    }

    return true;
}
```

6.2 First True

```
/*
  first_true(2, 10, [](int x) { return x * x >= 30;
    }); // outputs 6

  [l, r]

  if none of the values in the range work, return hi
    + 1

  f(1) = false
  f(2) = false
  f(3) = false
  f(4) = false
  f(5) = false
  f(6) = true
  f(7) = true
  f(8) = true
*/

int first_true(int lo, int hi, function<bool(int)> f)
     {
    hi++;
    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;

        if (f(mid)) {
            hi = mid;
        } else {
            lo = mid + 1;
        }
    }
    return lo;
}
```

6.3 Random

```
random_device dev;
mt19937 rng(dev());

uniform_int_distribution<mt19937::result_type> dist
    (1, 6); // distribution in range [1, 6]

int val = dist(rng);
```

6.4 Bitwise

```
#define lcm(a,b) (a*b)/gcd(a,b)
#define msb(n) (32 - __builtin_clz(n))
```

6.5 Min Priority Queue

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

6.6 Input By File

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

6.7 Read

```
/*
    RELER O ENUNCIADO!
    WA? coloca long long que passa;
    testar casos de borda, n = 0? n = 1? todos os
    numeros iguais?
    Uma resposta Ãştima pode ter tamanho 2?
    pode ser DP? nÃčo ÃĽ guloso de mais?
    dÃą pra modelar como grafo?
    pode ser fluxo?
    as vezes compensa mais fazer um brute (em python)

    nada funcionou? coda do zero
*/
```

6.8 Compilation Flags

```
/*
    // josÃĽ
    g++ -std=c++17 -Wshadow -Wall -Wextra -Wformat=2
    -Wconversion -fsanitize=address,undefined -fno-
    sanitize-recover -Wfatal-errors

    // maxwell

*/
```

6.9 Xor 1 To N

```
// XOR sum from 1 to N
ll xor_1_to_n(ll n) {
    if (n % 4 == 0) {
        return n;
    } else if (n % 4 == 1) {
        return 1;
    } else if (n % 4 == 2) {
        return n + 1;
    }

    return 0;
}
```

6.10 Next Permutation

```
// output: 1,2,3; 1,3,2; 2,1,3; 2,3,1; 3,1,2; 3,2,1;

vector<int> arr = {1, 2, 3};
int n = arr.size();

do {
    for (auto e : arr) {
        cout << e << ' ';
    }
    cout << '\n';
} while (next_permutation(arr.begin(), arr.end()));
```

6.11 Template Full

```cpp
#include <bits/stdc++.h>
#define debug(x) cout << "[" << #x << " = " << x << " ] "
#define ff first
#define ss second

using namespace std;
using ll = long long;
using ld = long double;
using pii = pair<int,int>;
using vi = vector<int>;

using tii = tuple<int,int,int>;

const int oo = (int)1e9 + 17; //INF to INT
const ll OO = 0x3f3f3f3f3f3f3f3fLL; //INF to LL

void solve(){

}

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

    int t = 1;
    cin >> t;
    while(t--){
        solve();
    }
}
```

6.12 Goto

```cpp
while (t--) {
  for (int d = 0; d < 11; d++) {
    if (n % 11 == 0) {
      cout << "YES" << endl;
      goto done;
    }

    n -= 111;
    if (n < 0) break;
  }
  cout << "NO" << endl;

  done:;
}
```

6.13 Split

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

    for (int i = 0; i < (int)s.size(); i++) {
        if (s[i] == key) {
            if (aux.size() > 0) {
                ans.push_back(aux);
                aux = "";
            }
        } else {
            aux += s[i];
        }
    }

    if ((int)aux.size() > 0) {
        ans.push_back(aux);
    }

    return ans;
}
```

6.14 Last True

```cpp
/*
  last_true(2, 10, [](int x) { return x * x <= 30; })
    ; // outputs 5

  [l, r]

  if none of the values in the range work, return lo
    - 1

  f(1) = true
  f(2) = true
  f(3) = true
  f(4) = true
  f(5) = true
  f(6) = false
  f(7) = false
  f(8) = false

  last_true(1, 8, f) = 5
  last_true(7, 8, f) = 6
*/

int last_true(int lo, int hi, function<bool(int)> f)
    {
    lo--;
    while (lo < hi) {
        int mid = lo + (hi - lo + 1) / 2;

        if (f(mid)) {
            lo = mid;
        } else {
            hi = mid - 1;
        }
    }
    return lo;
}
```

6.15 Template

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

using namespace std;

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



    return 0;
}
```

6.16 Ordered Set

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

using namespace __gnu_pbds;

typedef tree<
    int,
    null_type,
    less<int>,
    rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;

void Erase(ordered_set& a, int x){
    int r = a.order_of_key(x);
    auto it = a.find_by_order(r);
    a.erase(it);
}
```

```

/*
    order_of_key(k) // Number of items strictly smaller than k.
    find_by_order(k) // K-th element in a set (counting from zero).
*/

/* os parametros sÃčo, na ordem:
1) int -> tipo do valor que quero inserir(key), pode ser int, double, pii e etc
2) null_type -> Ãĺ para usar essa Ãąrvore como set/multiset. DÃą pra usar essa ED como map tmb trocando isso pra um tipo map
3) less<int> -> forma de comparaÃğÃčo dos elementos. less<int>, less_equal<int>, greater, greater_equal e etc
4) rb_tree_tag -> tipo de Ãąrvore a ser usado, usar a rb para as operaÃğÃţes serem O(logN)
5) tree_order_statistics_node__update -> contÃľm vÃąrias operaÃğÃţes para atualizar "tree-based container", usado pra manter algo como o numero de nodes em alguma subÃąrvore

usar less<equal> em (3) para usar como set e less_equal<TIPO> para usar como multiset (permitir elementos repetidos).
se em (3) colocar greater_equal, entao o order_of_key(k) retorna a quantidade de valores maiores que k (Ãĺ preferivel usar less_equal e retornar o tamanho do intervalo - order)
*/

/* Como usar
 .find_by_order(k) -> retorna um ITERADOR para o elemento na posiÃğÃčo kth (contando do 0, ou seja, k = 0 retorna o menor)
    Usar *(find_by_order(k)) para saber QUAL Ãĺ o numero na posiÃğÃčo.

 .order_of_key(k) -> retorna o numero de valores na Ãąrvore que estÃčo ESTRITAMENTE menores que k; retorna um inteiro

 TambÃľm Ãĺ possivel usar as funÃğÃţes que o set comum possui, como o .insert().
 DÃą pra percorrer os valores inseridos na Ãąrvore usando o for(auto p: tree);
 Obs.: O .erase funciona, mas precisa de um iterador para o elemento! Para isso tem a funÃğÃčo Erase implementada.
*/
```

6.17 Get Subset Sums

```
using ll = long long;

vector<ll> get_subset_sums(int l, int r, vector<ll> &arr) {
    vector<ll> ans;

    int len = r-l+1;
    for (int i = 0; i < (1 << len); i++) {
        ll sum = 0;

        for (int j = 0; j < len; j++) {
            if (i&(1 << j)) {
                sum += arr[l + j];
            }
        }

        ans.push_back(sum);
    }

    return ans;
}
```

7 DS

7.1 Segtree2d

```
// Description:
// Indexed at zero
// Given a N x M grid, where i represents the row and j the column, perform the following operations
// update(j, i) - update the value of grid[i][j]
// query(j1, j2, i1, i2) - return the sum of values inside the rectangle
// defined by grid[i1][j1] and grid[i2][j2] inclusive

// Problem:
// https://cses.fi/problemset/task/1739/

// Complexity:
// Time complexity:
// O(log N * log M) for both query and update
// O(N * M) for build
// Memory complexity:
// 4 * M * N

// How to use:
// Segtree2D seg = Segtree2D(n, n);
// vector<vector<int>> v(n, vector<int>(n));
// seg.build(v);

// Notes
// Indexed at zero

struct Segtree2D {
    const int MAXN = 1025;
    int N, M;

    vector<vector<int>> seg;

    Segtree2D(int N, int M) {
        this->N = N;
        this->M = M;
        seg.resize(2*MAXN, vector<int>(2*MAXN));
    }

    void buildY(int noX, int lX, int rX, int noY, int lY, int rY, vector<vector<int>> &v){
        if(lY == rY){
            if(lX == rX){
                seg[noX][noY] = v[rX][rY];
            }else{
                seg[noX][noY] = seg[2*noX+1][noY] + seg[2*noX+2][noY];
            }
        }else{
            int m = (lY+rY)/2;

            buildY(noX, lX, rX, 2*noY+1, lY, m, v);
            buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);

            seg[noX][noY] = seg[noX][2*noY+1] + seg[noX][2*noY+2];
        }
    }

    void buildX(int noX, int lX, int rX, vector<vector<int>> &v){
        if(lX != rX){
            int m = (lX+rX)/2;

```

```
            buildX(2*noX+1, lX, m, v);
            buildX(2*noX+2, m+1, rX, v);
        }

        buildY(noX, lX, rX, 0, 0, M - 1, v);
    }

    void updateY(int noX, int lX, int rX, int noY, int lY, int rY, int y){
        if(lY == rY){
            if(lX == rX){
                seg[noX][noY] = !seg[noX][noY];
            }else{
                seg[noX][noY] = seg[2*noX+1][noY] + seg[2*noX+2][noY];
            }
        }else{
            int m = (lY+rY)/2;

            if(y <= m){
                updateY(noX, lX, rX, 2*noY+1,lY, m, y);
            }else if(m < y){
                updateY(noX, lX, rX, 2*noY+2, m+1, rY, y);
            }

            seg[noX][noY] = seg[noX][2*noY+1] + seg[noX][2*noY+2];
        }
    }

    void updateX(int noX, int lX, int rX, int x, int y){
        int m = (lX+rX)/2;

        if(lX != rX){
            if(x <= m){
                updateX(2*noX+1, lX, m, x, y);
            }else if(m < x){
                updateX(2*noX+2, m+1, rX, x, y);
            }
        }

        updateY(noX, lX, rX, 0, 0, M - 1, y);
    }

    int queryY(int noX, int noY, int lY, int rY, int aY, int bY){
        if(aY <= lY && rY <= bY) return seg[noX][noY];

        int m = (lY+rY)/2;

        if(bY <= m) return queryY(noX, 2*noY+1, lY, m, aY, bY);
        if(m < aY) return queryY(noX, 2*noY+2, m+1, rY, aY, bY);

        return queryY(noX, 2*noY+1, lY, m, aY, bY) + queryY(noX, 2*noY+2, m+1, rY, aY, bY);
    }

    int queryX(int noX, int lX, int rX, int aX, int bX, int aY, int bY){
        if(aX <= lX && rX <= bX) return queryY(noX, 0, 0, M - 1, aY, bY);

        int m = (lX+rX)/2;

        if(bX <= m) return queryX(2*noX+1, lX, m, aX, bX, aY, bY);
        if(m < aX) return queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);

        return queryX(2*noX+1, lX, m, aX, bX, aY, bY) + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
    }

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

    int query(int aX, int bX, int aY, int bY) {
        return queryX(0, 0, N - 1, aX, bX, aY, bY);
    }

    void update(int x, int y) {
        updateX(0, 0, N - 1, x, y);
    }
};
```

7.2 Minimum And Amount

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

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

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

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

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

const int INF = 1e9+17;

typedef pii ftype;

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

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

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

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

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

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

```
        }

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

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

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

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

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

            return;
        }

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

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

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

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

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

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

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

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

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

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

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

7.3 Segment With Maximum Sum

```
// Description:
// Query - get sum of segment that is maximum among all segments
// E.g
// Array: 5 -4 4 3 -5
// Maximum segment sum: 8 because 5 + (-4) + 4 + 3 = 8
// Update - update element at position id to a value val

// Problem:
// https://codeforces.com/edu/course/2/lesson/4/2/practice/contest/273278/problem/A

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

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

// Notes
// The maximum segment sum can be a negative number
// In that case, taking zero elements is the best choice
// So we need to take the maximum between 0 and the query
// max(0LL, seg.query(0, n).max_seg)

using ll = long long;

typedef ll ftype_node;

struct Node {
    ftype_node max_seg;
    ftype_node pref;
    ftype_node suf;
    ftype_node sum;

    Node(ftype_node max_seg, ftype_node pref, ftype_node suf, ftype_node sum) : max_seg(max_seg), pref(pref), suf(suf), sum(sum) {};
};

typedef Node ftype;

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

    Segtree(int n) {
        int sz = 1;
        // potencia de dois mais proxima
        while (sz < n) sz *= 2;
        this->n = sz;

        // numero de nos da seg
        seg.assign(2*sz, NEUTRAL);
    }

    ftype f(ftype a, ftype b) {
        ftype_node max_seg = max({a.max_seg, b.max_seg, a.suf + b.pref});
        ftype_node pref = max(a.pref, a.sum + b.pref);
        ftype_node suf = max(b.suf, b.sum + a.suf);
        ftype_node sum = a.sum + b.sum;

        return Node(max_seg, pref, suf, sum);
```

```
    }

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

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

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

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

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

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

            return;
        }

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

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

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

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            // se a posiÃğÃčo existir no array original
            // seg tamanho potencia de dois
            if (ini < (int)v.size()) {
                seg[pos] = Node(v[ini], v[ini], v[ini], v[ini]);
            }
            return;
        }

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

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

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

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

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

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

    void debug() {
        for (auto e : seg) {
            cout << e.max_seg << ' ' << e.pref << ' ' << e.suf << ' ' << e.sum << '\n';
        }
        cout << '\n';
    }
};
```

7.4 Persistent

```
// Description:
// Persistent segtree allows for you to save the different versions of the segtree between each update
// Indexed at one
// Query - get sum of elements from range (l, r) inclusive
// Update - update element at position id to a value val

// Problem:
// https://cses.fi/problemset/task/1737/

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

// How to use:
// vector<int> raiz(MAX); // vector to store the roots of each version
// Segtree seg = Segtree(INF);
// raiz[0] = seg.create(); // null node
// curr = 1; // keep track of the last version

// raiz[k] = seg.update(raiz[k], idx, val); // updating version k
// seg.query(raiz[k], l, r) // querying version k
// raiz[++curr] = raiz[k]; // create a new version based on version k

const int MAX = 2e5+17;
const int INF = 1e9+17;

typedef long long ftype;

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

    Segtree(int n) {
        this->n = n;
    }

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

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
```

```
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

    int update(int pos, int ini, int fim, int id, int val) {
        int novo = create();

        seg[novo] = seg[pos];
        e[novo] = e[pos];
        d[novo] = d[pos];

        if (ini == fim) {
            seg[novo] = val;
            return novo;
        }

        int m = (ini + fim) >> 1;

        if (id <= m) e[novo] = update(e[novo], ini, m, id, val);
        else d[novo] = update(d[novo], m + 1, fim, id, val);

        seg[novo] = f(seg[e[novo]], seg[d[novo]]);

        return novo;
    }

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

    int update(int pos, int id, int val) {
        return update(pos, 1, n, id, val);
    }
};
```

7.5 Range Query Point Update

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

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

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

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

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

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

typedef long long ftype;

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

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

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

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

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

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

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

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

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

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

            return;
        }

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

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

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

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

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

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

```

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

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

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

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

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

7.6 Lazy Assignment To Segment

```
const long long INF = 1e18+10;

typedef long long ftype;

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

    Segtree(int n) {
        int sz = 1;
        // potencia de dois mais proxima
        while (sz < n) sz *= 2;
        this->n = sz;

        // numero de nos da seg
        seg.assign(2*sz, NEUTRAL);
        lazy.assign(2*sz, NEUTRAL_LAZY);
    }

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

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

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

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

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

        lazy[pos] = NEUTRAL_LAZY;
    }

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

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

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

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

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

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

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

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

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

            return;
        }

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

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

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

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            // se a posiÃğÃčo existir no array original
            // seg tamanho potencia de dois
            if (ini < (int)v.size()) {
                seg[pos] = v[ini];
            }
            return;
        }

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

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

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

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

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

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

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

7.7 Dynamic Implicit Sparse

```
// Description:
// Indexed at one

// When the indexes of the nodes are too big to be stored in an array
// and the queries need to be answered online so we can't sort the nodes and compress them
// we create nodes only when they are needed so there'll be (Q*log(MAX)) nodes
// where Q is the number of queries and MAX is the maximum index a node can assume

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

// Problem:
// https://cses.fi/problemset/task/1648

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

// How to use:
// MAX is the maximum index a node can assume

// Segtree seg = Segtree(MAX);

typedef long long ftype;

const int MAX = 1e9+17;

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

    Segtree(int n) {
        this->n = n;
        create();
        create();
    }

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

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

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

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

            return;
        }

        int m = (ini + fim) >> 1;

        if (id <= m) {
            if (e[pos] == 0) e[pos] = create();
            update(e[pos], ini, m, id, val);
        } else {
            if (d[pos] == 0) d[pos] = create();
            update(d[pos], m + 1, fim, id, val);
        }

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

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

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

7.8 Lazy Addition To Segment

```
// Description:
// Query - get sum of elements from range (l, r) inclusive
// Update - add a value val to elementos from range (l, r) inclusive

// Problem:
// https://codeforces.com/edu/course/2/lesson/5/1/practice/contest/279634/problem/A

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

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

// Notes
```

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

const long long INF = 1e18+10;

typedef long long ftype;

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

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

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

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

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

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

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

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

        lazy[pos] = NEUTRAL_LAZY;
    }

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

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

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

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

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

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

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

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

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

            return;
        }

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

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

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

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

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

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

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

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

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

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

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

7.9 Lazy Dynamic Implicit Sparse

```cpp
// Description:
// Indexed at one

// When the indexes of the nodes are too big to be stored in an array
// and the queries need to be answered online so we can't sort the nodes and compress them
// we create nodes only when they are needed so there'll be (Q*log(MAX)) nodes
// where Q is the number of queries and MAX is the maximum index a node can assume

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

// Problem:
// https://oj.uz/problem/view/IZhO12_apple

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

// How to use:
// MAX is the maximum index a node can assume
// Create a default null node
// Create a node to be the root of the segtree

// Segtree seg = Segtree(MAX);
// seg.create();
// seg.create();

const int MAX = 1e9+10;
const long long INF = 1e18+10;

typedef long long ftype;

struct Segtree {
    vector<ftype> seg, d, e, lazy;
    const ftype NEUTRAL = 0;
    const ftype NEUTRAL_LAZY = -INF;
    int n;

    Segtree(int n) {
        this->n = n;
    }

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

    void propagate(int pos, int ini, int fim) {
        if (seg[pos] == 0) return;

        if (ini == fim) {
            return;
        }

        int m = (ini + fim) >> 1;

        if (e[pos] == 0) e[pos] = create();
        if (d[pos] == 0) d[pos] = create();

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

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

        lazy[pos] = NEUTRAL_LAZY;
    }

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

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        lazy.push_back(-1);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        propagate(pos, ini, fim);
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

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

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

            return;
        }

        int m = (ini + fim) >> 1;

        if (e[pos] == 0) e[pos] = create();
        update(e[pos], ini, m, p, q, val);

        if (d[pos] == 0) d[pos] = create();
        update(d[pos], m + 1, fim, p, q, val);

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

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

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

7.10 Big K

```cpp
struct SetSum {
    ll s = 0;
    multiset<ll> mt;
    void add(ll x){
        mt.insert(x);
        s += x;
    }
    int pop(ll x){
```

```
        auto f = mt.find(x);
        if(f == mt.end()) return 0;
        mt.erase(f);
        s -= x;
        return 1;
    }
};

struct BigK {
    int k;
    SetSum gt, mt;
    BigK(int _k){
        k = _k;
    }
    void balancear(){
        while((int)gt.mt.size() < k && (int)mt.mt.size()){
            auto p = (prev(mt.mt.end()));
            gt.add(*p);
            mt.pop(*p);
        }
        while((int)mt.mt.size() && (int)gt.mt.size() &&
        *(gt.mt.begin()) < *(prev(mt.mt.end())) ){
            ll u = *(gt.mt.begin());
            ll v = *(prev(mt.mt.end()));
            gt.pop(u); mt.pop(v);
            gt.add(v); mt.add(u);
        }
    }
    void add(ll x){
        mt.add(x);
        balancear();
    }
    void rem(ll x){
        //x = -x;
        if(mt.pop(x) == 0)
            gt.pop(x);
        balancear();
    }
};
```

7.11 Ordered Set

```
// Ordered Set
//
// set roubado com mais operacoes
//
// para alterar para multiset
// trocar less para less_equal
//
// ordered_set<int> s
//
// order_of_key(k) // number of items strictly smaller than k -> int
// find_by_order(k) // k-th element in a set (counting from zero) -> iterator
//
// https://cses.fi/problemset/task/2169
//
// O(log N) para insert, erase (com iterator), order_of_key, find_by_order

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

7.12 Dsu

```
/*
DSU - Disjoint Set Union (or Union Find)

find(x) -> find component that x is on
join(a, b) -> union of a set containing 'a' and set containing b

find / join with path compreension -> O(inv_Ackermann(n)) [O(1)]
find / join without path compreension -> O(logN)

https://judge.yosupo.jp/submission/126864
*/

struct DSU {

    int n = 0, components = 0;
    vector<int> parent;
    vector<int> size;

    DSU(int nn){
        n = nn;
        components = n;
        size.assign(n + 5, 1);
        parent.assign(n + 5, 0);
        iota(parent.begin(), parent.end(), 0);
    }

    int find(int x){
        if(x == parent[x]) {
            return x;
        }
        //path compression
        return parent[x] = find(parent[x]);
    }

    void join(int a, int b){
        a = find(a);
        b = find(b);
        if(a == b) {
            return;
        }
        if(size[a] < size[b]) {
            swap(a, b);
        }
        parent[b] = a;
        size[a] += size[b];
        components -= 1;
    }

    int sameSet(int a, int b) {
        a = find(a);
        b = find(b);
        return a == b;
    }

};
```