



Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1 Math	2	3 Template	7
1.1 Ceil	2	3.1 Template	7
1.2 To Decimal	2	3.2 Template Clean	7
1.3 Subsets	2	4 Strings	8
1.4 Matrix Exponentiation	2	4.1 Kmp	8
1.5 Crt	3	4.2 Generate All Permutations	8
1.6 Binary To Decimal	3	4.3 Generate All Sequences Length K	8
1.7 Fast Exponentiation	3	4.4 Lcs	8
1.8 Linear Diophantine Equation	3	4.5 Trie	8
1.9 Function Root	4	4.6 Z-function	9
1.10 Sieve Of Eratosthenes	4	5 Misc	9
1.11 Horner Algorithm	4	5.1 Split	9
1.12 Multiplicative Inverse	4	5.2 Int128	9
1.13 Representation Arbitrary Base	5	6 Graphs	9
1.14 Set Operations	5	6.1 Centroid Find	9
1.15 Divisors	5	6.2 Bipartite	10
1.16 Check If Bit Is On	5	6.3 Prim	10
1.17 Prime Factors	5	6.4 Ford Fulkerson Edmonds Karp	10
2 DP	5	6.5 Floyd Warshall	11
2.1 Knapsack With Index	5	6.6 Lca	11
2.2 Substr Palindrome	6	6.7 Bellman Ford	12
2.3 Edit Distance	6	6.8 Dinic	12
2.4 Knapsack	6	6.9 2sat	14
2.5 Digits	6	6.10 Find Cycle	15
2.6 Coins	6	6.11 Hld	15
2.7 Minimum Coin Change	7	6.12 Cycle Path Recovery	17
2.8 Kadane	7	6.13 Centroid Decomposition	17
		6.14 Tarjan Bridge	17
		6.15 Small To Large	18
		6.16 Tree Diameter	18

6.17	Dijkstra	19
6.18	Kruskall	19
7	Geometry	20
7.1	2d	20
8	Algorithms	22
8.1	Lis	22
8.2	Delta-encoding	22
8.3	Subsets	22
8.4	Binary Search Last True	22
8.5	Ternary Search	22
8.6	Binary Search First True	23
8.7	Biggest K	23
9	Data Structures	23
9.1	Ordered Set	23
9.2	Priority Queue	24
9.3	Dsu	24
9.4	Two Sets	24
9.5	Dynamic Implicit Sparse	25
9.6	Segtree2d	26
9.7	Minimum And Amount	27
9.8	Lazy Addition To Segment	28
9.9	Segment With Maximum Sum	29
9.10	Range Query Point Update	30
9.11	Lazy Assignment To Segment	31
9.12	Lazy Dynamic Implicit Sparse	32
9.13	Persistent	33

1 Math

1.1 Ceil

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

1.2 To Decimal

```
1 const string digits { "0123456789
2     ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
3
4 long long to_decimal(const string& rep, long long
5     base) {
6     long long n = 0;
7
8     for (auto c : rep) {
9         // if the number can't be represented in this
10        base
11        if (c > digits[base - 1]) return -1;
12        n *= base;
13        n += digits.find(c);
14    }
15
16    return n;
17 }
```

1.3 Subsets

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

1.4 Matrix Exponentiation

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

```
21
22 // Adding another column, there are:
23 // 3 ways to go from 2 equal to 2 equal
24 // 2 ways to go from 2 equal to 3 distinct
25 // 2 ways to go from 3 distinct to 2 equal
26 // 2 ways to go from 3 distinct to 3 distinct
27
28 // So we start with matrix 6 6 and multiply it by the
29 // transition 3 2 and get 18 12
30 //
31 //          2 2          6 6          12 12
32 // the we can exponentiate this matrix to find the
33 // nth column
34
35 // Problem:
36 // https://cses.fi/problemset/task/1722/
37
38 // Complexity:
39 // O(log n)
40
41 // How to use:
42 // vector<vector<ll>> v = {{1, 1}, {1, 0}};
43 // Matriz transition = Matriz(v);
44 // cout << fexp(transition, n)[0][1] << '\n';
45
46 using ll = long long;
47
48 const int MOD = 1e9+7;
49
50 struct Matriz{
51     vector<vector<ll>> mat;
52     int rows, columns;
53
54     vector<ll> operator[](int i){
55         return mat[i];
56     }
57
58     Matriz(vector<vector<ll>>& matriz){
59         mat = matriz;
60         rows = mat.size();
61         columns = mat[0].size();
62     }
63
64     Matriz(int row, int column, bool identity=false){
65         rows = row; columns = column;
66         mat.assign(rows, vector<ll>(columns, 0));
67         if(identity) {
68             for(int i = 0; i < min(rows, columns); i
69             ++){
70                 mat[i][i] = 1;
71             }
72         }
73     }
74
75     Matriz operator * (Matriz a) {
76         assert(columns == a.rows);
77         vector<vector<ll>> resp(rows, vector<ll>(a.
78         columns, 0));
79
80         for(int i = 0; i < rows; i++){
81             for(int j = 0; j < a.columns; j++){
82                 for(int k = 0; k < a.rows; k++){
83                     resp[i][j] = (resp[i][j] + (mat[i
84                     ][k] * 1LL * a[k][j]) % MOD) % MOD;
85                 }
86             }
87         }
88         return Matriz(resp);
89     }
90
91     Matriz operator + (Matriz a) {
92         assert(rows == a.rows && columns == a.columns
93         );
```

```

87     vector<vector<ll>> resp(rows, vector<ll>(
columns,0));
88     for(int i = 0; i < rows; i++){
89         for(int j = 0; j < columns; j++){
90             resp[i][j] = (resp[i][j] + mat[i][j]
+ a[i][j]) % MOD;
91         }
92     }
93     return Matriz(resp);
94 }
95 };
96
97 Matriz fexp(Matriz base, ll exponent){
98     Matriz result = Matriz(base.rows, base.rows, 1);
99     while(exponent > 0){
100         if(exponent & 1LL) result = result * base;
101         base = base * base;
102         exponent = exponent >> 1;
103     }
104     return result;
105 }

```

1.5 Crt

```

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

```

1.6 Binary To Decimal

```

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

```

1.7 Fast Exponentiation

```

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

```

1.8 Linear Diophantine Equation

```

1 // int a, b, c, x1, x2, y1, y2; cin >> a >> b >> c >>
x1 >> x2 >> y1 >> y2;
2 // int ans = -1;
3 // if (a == 0 && b == 0) {
4 //     if (c != 0) ans = 0;
5 //     else ans = (x2 - x1 + 1) * (y2 - y1 + 1);
6 // }
7 // else if (a == 0) {
8 //     if (c % b == 0 && y1 <= c / b && y2 >= c / b)
ans = (x2 - x1 + 1);
9 //     else ans = 0;
10 // }
11 // else if (b == 0) {
12 //     if (c % a == 0 && x1 <= c / a && x2 >= c / a)
ans = (y2 - y1 + 1);
13 //     else ans = 0;
14 // }
15
16 // Careful when a or b are negative or zero
17
18 // if (ans == -1) ans = find_all_solutions(a, b, c,
x1, x2, y1, y2);
19 // cout << ans << '\n';
20
21 // Problems:
22 // https://www.spoj.com/problems/CEQU/
23 // http://codeforces.com/problemsets/acmsguru/problem
/99999/106
24
25 // consider trivial case a or b is 0
26 int gcd(int a, int b, int& x, int& y) {
27     if (b == 0) {
28         x = 1;
29         y = 0;
30         return a;
31     }
32     int x1, y1;
33     int d = gcd(b, a % b, x1, y1);
34     x = y1;
35     y = x1 - y1 * (a / b);
36     return d;
37 }
38
39 // x and y are one solution and g is the gcd, all
passed as reference
40 // minx <= x <= maxx miny <= y <= maxy
41 bool find_any_solution(int a, int b, int c, int &x0,
int &y0, int &g) {
42     g = gcd(abs(a), abs(b), x0, y0);
43     if (c % g) {
44         return false;
45     }
46
47     x0 *= c / g;
48     y0 *= c / g;
49     if (a < 0) x0 = -x0;
50     if (b < 0) y0 = -y0;
51     return true;
52 }

```

```

53
54 void shift_solution(int & x, int & y, int a, int b,
    int cnt) {
55     x += cnt * b;
56     y -= cnt * a;
57 }
58
59 // return number of solutions in the interval
60 int find_all_solutions(int a, int b, int c, int minx,
    int maxx, int miny, int maxy) {
61     int x, y, g;
62     if (!find_any_solution(a, b, c, x, y, g))
63         return 0;
64     a /= g;
65     b /= g;
66
67     int sign_a = a > 0 ? +1 : -1;
68     int sign_b = b > 0 ? +1 : -1;
69
70     shift_solution(x, y, a, b, (minx - x) / b);
71     if (x < minx)
72         shift_solution(x, y, a, b, sign_b);
73     if (x > maxx)
74         return 0;
75     int lx1 = x;
76
77     shift_solution(x, y, a, b, (maxx - x) / b);
78     if (x > maxx)
79         shift_solution(x, y, a, b, -sign_b);
80     int rx1 = x;
81
82     shift_solution(x, y, a, b, -(miny - y) / a);
83     if (y < miny)
84         shift_solution(x, y, a, b, -sign_a);
85     if (y > maxy)
86         return 0;
87     int lx2 = x;
88
89     shift_solution(x, y, a, b, -(maxy - y) / a);
90     if (y > maxy)
91         shift_solution(x, y, a, b, sign_a);
92     int rx2 = x;
93
94     if (lx2 > rx2)
95         swap(lx2, rx2);
96     int lx = max(lx1, lx2);
97     int rx = min(rx1, rx2);
98
99     if (lx > rx)
100         return 0;
101     return (rx - lx) / abs(b) + 1;
102 }

```

1.9 Function Root

```

1 const ld EPS1 = 1e-9; // iteration precision error
2 const ld EPS2 = 1e-4; // output precision error
3
4 ld f(ld x) {
5     // exp(-x) == e^(-x)
6     return p * exp(-x) + q * sin(x) + r * cos(x) + s *
    tan(x) + t * x * x + u;
7 }
8
9 ld root(ld a, ld b) {
10     while (b - a >= EPS1) {
11         ld c = (a + b) / 2.0;
12         ld y = f(c);
13
14         if (y < 0) b = c;
15         else a = c;
16     }
17 }

```

```

18     return (a + b) / 2;
19 }
20
21 int main() {
22     ld ans = root(0, 1);
23     if (abs(f(ans)) <= EPS2) cout << fixed <<
        setprecision(4) << ans << '\n';
24     else cout << "No solution\n";
25
26     return 0;
27 }

```

1.10 Sieve Of Eratosthenes

```

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

```

1.11 Horner Algorithm

```

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

```

1.12 Multiplicative Inverse

```

1 ll extend_euclid(ll a, ll b, ll &x, ll &y) {
2     if (a == 0)
3     {
4         x = 0; y = 1;
5         return b;
6     }
7     ll x1, y1;
8     ll d = extend_euclid(b%a, a, x1, y1);
9     x = y1 - (b / a) * x1;

```

```

10     y = x1;
11     return d;
12 }
13
14 // gcd(a, m) = 1 para existir solucao
15 // ax + my = 1, ou a*x = 1 (mod m)
16 ll inv_gcd(ll a, ll m) { // com gcd
17     ll x, y;
18     extend_euclid(a, m, x, y);
19     return ((x % m) + m) % m;
20 }
21
22 ll inv(ll a, ll phim) { // com phi(m), se m for primo
23     entao phi(m) = p-1
24     ll e = phim-1;
25     return fexp(a, e, MOD);
26 }

```

1.13 Representation Arbitrary Base

```

1 const string digits { "0123456789
2     ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
3
4 string representation(int n, int b) {
5     string rep;
6
7     do {
8         rep.push_back(digits[n % b]);
9         n /= b;
10    } while (n);
11
12    reverse(rep.begin(), rep.end());
13
14    return rep;
15 }

```

1.14 Set Operations

```

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

```

1.15 Divisors

```

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

```

1.16 Check If Bit Is On

```

1 // msb de 0 é undefined
2 #define msb(n) (32 - __builtin_clz(n))
3 // #define msb(n) (64 - __builtin_clzll(n) )
4 // popcount
5 // turn bit off
6
7 bool bit_on(int n, int bit) {
8     if(1 & (n >> bit)) return true;
9     else return false;
10 }

```

1.17 Prime Factors

```

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

```

2 DP

2.1 Knapsack With Index

```

1 void knapsack(int W, int wt[], int val[], int n) {
2     int i, w;
3     int K[n + 1][W + 1];
4
5     for (i = 0; i <= n; i++) {
6         for (w = 0; w <= W; w++) {
7             if (i == 0 || w == 0)
8                 K[i][w] = 0;
9             else if (wt[i - 1] <= w)
10                K[i][w] = max(val[i - 1] +
11                    K[i - 1][w - wt[i - 1]], K[i -
12                1][w]);
13            else
14                K[i][w] = K[i - 1][w];
15        }
16    }
17
18    int res = K[n][W];
19    cout << res << endl;
20
21    w = W;
22    for (i = n; i > 0 && res > 0; i--) {
23        if (res == K[i - 1][w])
24            continue;
25        else {
26            cout << " " << wt[i - 1] ;
27            res = res - val[i - 1];
28            w = w - wt[i - 1];
29        }
30    }
31
32    int main()
33    {
34        int val[] = { 60, 100, 120 };
35        int wt[] = { 10, 20, 30 };
36        int W = 50;
37        int n = sizeof(val) / sizeof(val[0]);

```

```

38     knapsack(W, wt, val, n);
39
40
41     return 0;
42 }

```

2.2 Substr Palindrome

```

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

```

2.3 Edit Distance

```

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

```

```

34     return dp[m][n] = 1 + min({edit_distance(str1,
  str2, m, n - 1), edit_distance(str1, str2, m - 1,
  n), edit_distance(str1, str2, m - 1, n - 1)});
35 }

```

2.4 Knapsack

```

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

```

2.5 Digits

```

1 // achar a quantidade de numeros menores que R que
  possuem no maximo 3 digitos nao nulos
2 // a ideia eh utilizar da ordem lexicografica para
  checar isso pois se temos por exemplo
3 // o numero 8500, a gente sabe que se pegarmos o
  numero 7... qualquer digito depois do 7
4 // sera necessariamente menor q 8500
5
6 string r;
7 int tab[20][2][5];
8
9 // i - digito de R
10 // menor - ja pegou um numero menor que um digito de
  R
11 // qt - quantidade de digitos nao nulos
12 int dp(int i, bool menor, int qt){
13     if(qt > 3) return 0;
14     if(i >= r.size()) return 1;
15     if(tab[i][menor][qt] != -1) return tab[i][menor][
  qt];
16
17     int dr = r[i]-'0';
18     int res = 0;
19
20     for(int d = 0; d <= 9; d++) {
21         int dnn = qt + (d > 0);
22         if(menor == true) {
23             res += dp(i+1, true, dnn);
24         }
25         else if(d < dr) {
26             res += dp(i+1, true, dnn);
27         }
28         else if(d == dr) {
29             res += dp(i+1, false, dnn);
30         }
31     }
32
33     return tab[i][menor][qt] = res;
34 }

```

2.6 Coins

```

1 int tb[1005];
2 int n;
3 vector<int> moedas;
4
5 int dp(int i){

```

```

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

```

2.7 Minimum Coin Change

```

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

```

2.8 Kadane

```

1 // achar uma subsequencia continua no array que a
2 // soma seja a maior possivel
3 // nesse caso vc precisa multiplicar exatamente 1
4 // elemento da subsequencia
5 // e achar a maior soma com isso
6
7 int n, x, arr[MAX], tab[MAX][2]; // tab[maior
8 // resposta no intervalo][foi multiplicado ou ão]
9
10 int dp(int i, bool mult) {
11     if (i == n-1) {
12         if (!mult) return arr[n-1]*x;
13         return arr[n-1];
14     }
15     if (tab[i][mult] != -1) return tab[i][mult];
16
17     int res;
18
19     if (mult) {
20         res = max(arr[i], arr[i] + dp(i+1, 1));
21     }
22     else {
23         res = max({
24             arr[i]*x,
25             arr[i]*x + dp(i+1, 1),
26             arr[i] + dp(i+1, 0)
27         });
28     }
29
30     return tab[i][mult] = res;
31 }
32
33 int main() {
34     memset(tab, -1, sizeof(tab));
35 }

```

```

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

```

3 Template

3.1 Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define int long long
5 #define optimize std::ios::sync_with_stdio(false);
6 cin.tie(NULL);
7
8 #define vi vector<int>
9 #define ll long long
10 #define pb push_back
11 #define mp make_pair
12 #define ff first
13 #define ss second
14 #define pii pair<int, int>
15 #define MOD 1000000007
16 #define sqr(x) ((x) * (x))
17 #define all(x) (x).begin(), (x).end()
18 #define FOR(i, j, n) for (int i = j; i < n; i++)
19 #define qle(i, n) (i == n ? "\n" : " ")
20 #define endl "\n"
21 const int oo = 1e9;
22 const int MAX = 1e6;
23
24 int32_t main(){ optimize;
25
26     return 0;
27 }

```

3.2 Template Clean

```

1 // Notes:
2 // Compile and execute
3 // g++ teste.cpp -o teste -std=c++17
4 // ./teste < teste.txt
5
6 // Print with precision
7 // cout << fixed << setprecision(12) << value << endl
8 // ;
9 // File as input and output
10 // freopen("input.txt", "r", stdin);
11 // freopen("output.txt", "w", stdout);

```



```

12
13 #include <bits/stdc++.h>
14 using namespace std;
15
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(NULL);
19
20
21
22     return 0;
23 }

```

4 Strings

4.1 Kmp

```

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

```

4.2 Generate All Permutations

```

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

```

4.3 Generate All Sequences Length K

```

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

```

4.4 Lcs

```

1 // Description:
2 // Finds the longest common subsequence between two
   string
3
4 // Problem:
5 // https://codeforces.com/gym/103134/problem/B
6
7 // Complexity:
8 // O(mn) where m and n are the length of the strings
9
10 string lcsAlgo(string s1, string s2, int m, int n) {
11     int LCS_table[m + 1][n + 1];
12
13     for (int i = 0; i <= m; i++) {
14         for (int j = 0; j <= n; j++) {
15             if (i == 0 || j == 0)
16                 LCS_table[i][j] = 0;
17             else if (s1[i - 1] == s2[j - 1])
18                 LCS_table[i][j] = LCS_table[i - 1][j - 1] +
19                     1;
20             else
21                 LCS_table[i][j] = max(LCS_table[i - 1][j],
22                     LCS_table[i][j - 1]);
23         }
24     }
25
26     int index = LCS_table[m][n];
27     char lcsAlgo[index + 1];
28     lcsAlgo[index] = '\0';
29
30     int i = m, j = n;
31     while (i > 0 && j > 0) {
32         if (s1[i - 1] == s2[j - 1]) {
33             lcsAlgo[index - 1] = s1[i - 1];
34             i--;
35             j--;
36             index--;
37         }
38         else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
39             i--;
40         else
41             j--;
42     }
43
44     return lcsAlgo;
45 }

```

4.5 Trie

```

1 const int K = 26;
2
3 struct Vertex {
4     int next[K];
5     bool output = false;
6     int p = -1;
7     char pch;
8     int link = -1;
9     int go[K];
10
11     Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
12         fill(begin(next), end(next), -1);
13         fill(begin(go), end(go), -1);
14     }
15 };
16
17 vector<Vertex> t(1);
18
19 void add_string(string const& s) {
20     int v = 0;
21     for (char ch : s) {
22         int c = ch - 'a';

```

```

23         if (t[v].next[c] == -1) {
24             t[v].next[c] = t.size();
25             t.emplace_back(v, ch);
26         }
27         v = t[v].next[c];
28     }
29     t[v].output = true;
30 }
31
32 int go(int v, char ch);
33
34 int get_link(int v) {
35     if (t[v].link == -1) {
36         if (v == 0 || t[v].p == 0)
37             t[v].link = 0;
38         else
39             t[v].link = go(get_link(t[v].p), t[v].pch
40     );
41     }
42     return t[v].link;
43 }
44
45 int go(int v, char ch) {
46     int c = ch - 'a';
47     if (t[v].go[c] == -1) {
48         if (t[v].next[c] != -1)
49             t[v].go[c] = t[v].next[c];
50         else
51             t[v].go[c] = v == 0 ? 0 : go(get_link(v),
52     ch);
53     }
54     return t[v].go[c];
55 }

```

4.6 Z-function

```

1 vector<int> z_function(string s) {
2     int n = (int) s.length();
3     vector<int> z(n);
4     for (int i = 1, l = 0, r = 0; i < n; ++i) {
5         if (i <= r)
6             z[i] = min(r - i + 1, z[i - l]);
7         while (i + z[i] < n && s[z[i]] == s[i + z[i]
8     ])
9             ++z[i];
10        if (i + z[i] - 1 > r)
11            l = i, r = i + z[i] - 1;
12    }
13    return z;
14 }

```

5 Misc

5.1 Split

```

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

```

```

17
18     if(palTemp.size() > 0)
19         ans.push_back(palTemp);
20
21     return ans;
22 }

```

5.2 Int128

```

1 __int128 read() {
2     __int128 x = 0, f = 1;
3     char ch = getchar();
4     while (ch < '0' || ch > '9') {
5         if (ch == '-') f = -1;
6         ch = getchar();
7     }
8     while (ch >= '0' && ch <= '9') {
9         x = x * 10 + ch - '0';
10        ch = getchar();
11    }
12    return x * f;
13 }
14
15 void print(__int128 x) {
16     if (x < 0) {
17         putchar('-');
18         x = -x;
19     }
20     if (x > 9) print(x / 10);
21     putchar(x % 10 + '0');
22 }

```

6 Graphs

6.1 Centroid Find

```

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

```

```

35     }
36     return node;
37 }
38
39 int main() {
40     cin >> n;
41     for (int i = 0; i < n - 1; i++) {
42         int u, v; cin >> u >> v;
43         u--; v--;
44         adj[u].push_back(v);
45         adj[v].push_back(u);
46     }
47
48     get_subtree_size(0);
49     cout << get_centroid(0) + 1 << endl;
50 }

```

6.2 Bipartite

```

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

```

6.3 Prim

```

1  int n;
2  vector<vector<int>> adj; // adjacency matrix of graph
3  const int INF = 1000000000; // weight INF means there
   is no edge
4
5  struct Edge {
6      int w = INF, to = -1;
7  };
8
9  void prim() {
10     int total_weight = 0;
11     vector<bool> selected(n, false);
12     vector<Edge> min_e(n);
13     min_e[0].w = 0;

```

```

14
15     for (int i=0; i<n; ++i) {
16         int v = -1;
17         for (int j = 0; j < n; ++j) {
18             if (!selected[j] && (v == -1 || min_e[j].
19                 w < min_e[v].w))
20                 v = j;
21
22         if (min_e[v].w == INF) {
23             cout << "No MST!" << endl;
24             exit(0);
25         }
26
27         selected[v] = true;
28         total_weight += min_e[v].w;
29         if (min_e[v].to != -1)
30             cout << v << " " << min_e[v].to << endl;
31
32         for (int to = 0; to < n; ++to) {
33             if (adj[v][to] < min_e[to].w)
34                 min_e[to] = {adj[v][to], v};
35         }
36     }
37
38     cout << total_weight << endl;
39 }

```

6.4 Ford Fulkerson Edmonds Karp

```

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

```

```

38     vector<int> parent(n);
39     int new_flow;
40
41     while (new_flow = bfs(s, t, parent)) {
42         flow += new_flow;
43         int cur = t;
44         while (cur != s) {
45             int prev = parent[cur];
46             capacity[prev][cur] -= new_flow;
47             capacity[cur][prev] += new_flow;
48             cur = prev;
49         }
50     }
51
52     return flow;
53 }

```

6.5 Floyd Warshall

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  using ll = long long;
5
6  const int MAX = 507;
7  const long long INF = 0x3f3f3f3f3f3f3fLL;
8
9  ll dist[MAX][MAX];
10 int n;
11
12 void floyd_warshall() {
13     for (int i = 0; i < n; i++) {
14         for (int j = 0; j < n; j++) {
15             if (i == j) dist[i][j] = 0;
16             else if (!dist[i][j]) dist[i][j] = INF;
17         }
18     }
19
20     for (int k = 0; k < n; k++) {
21         for (int i = 0; i < n; i++) {
22             for (int j = 0; j < n; j++) {
23                 // trata o caso no qual o grafo tem
24                 // arestas com peso negativo
25                 if (dist[i][k] < INF && dist[k][j] <
26                     INF){
27                     dist[i][j] = min(dist[i][j], dist
28                                     [i][k] + dist[k][j]);
29                 }
30             }
31         }
32     }
33 }

```

6.6 Lca

```

1  // Description:
2  // Find the lowest common ancestor between two nodes
3  // in a tree
4
5  // Problem:
6  // https://cses.fi/problemset/task/1135
7
8  // Complexity:
9  // O(log n)
10
11 // How to use:
12 // preprocess();
13 // lca(a, b);
14
15 // Notes
16 // To calculate the distance between two nodes use
17 // the following formula

```

```

16 // level_peso[a] + level_peso[b] - 2*level_peso[lca(a
17 // , b)]
18
19 const int MAX = 2e5+10;
20 const int BITS = 30;
21
22 vector<pii> adj[MAX];
23 vector<bool> visited(MAX);
24
25 int up[MAX][BITS + 1];
26 int level[MAX];
27 int level_peso[MAX];
28
29 void find_level() {
30     queue<pii> q;
31
32     q.push(mp(1, 0));
33     visited[1] = true;
34
35     while (!q.empty()) {
36         auto [v, depth] = q.front();
37         q.pop();
38         level[v] = depth;
39
40         for (auto [u, d] : adj[v]) {
41             if (!visited[u]) {
42                 visited[u] = true;
43                 up[u][0] = v;
44                 q.push(mp(u, depth + 1));
45             }
46         }
47     }
48 }
49
50 void find_level_peso() {
51     queue<pii> q;
52
53     q.push(mp(1, 0));
54     visited[1] = true;
55
56     while (!q.empty()) {
57         auto [v, depth] = q.front();
58         q.pop();
59         level_peso[v] = depth;
60
61         for (auto [u, d] : adj[v]) {
62             if (!visited[u]) {
63                 visited[u] = true;
64                 up[u][0] = v;
65                 q.push(mp(u, depth + d));
66             }
67         }
68     }
69 }
70
71 int lca(int a, int b) {
72     // get the nodes to the same level
73     int mn = min(level[a], level[b]);
74
75     for (int j = 0; j <= BITS; j++) {
76         if (a != -1 && ((level[a] - mn) & (1 << j))) a
77             = up[a][j];
78         if (b != -1 && ((level[b] - mn) & (1 << j))) b
79             = up[b][j];
80     }
81
82     // special case
83     if (a == b) return a;
84
85     // binary search
86     for (int j = BITS; j >= 0; j--) {
87         if (up[a][j] != up[b][j]) {
88             a = up[a][j];
89         }
90     }
91     return a;
92 }

```

```

86     b = up[b][j];
87 }
88 }
89 return up[a][0];
90 }
91
92 void preprocess() {
93     visited = vector<bool>(MAX, false);
94     find_level();
95     visited = vector<bool>(MAX, false);
96     find_level_peso();
97
98     for (int j = 1; j <= BITS; j++) {
99         for (int i = 1; i <= n; i++) {
100             if (up[i][j - 1] != -1) up[i][j] = up[up[i][j - 1]][j - 1];
101         }
102     }
103 }

```

6.7 Bellman Ford

```

1 struct edge
2 {
3     int a, b, cost;
4 };
5
6 int n, m, v;
7 vector<edge> e;
8 const int INF = 1000000000;
9
10 void solve()
11 {
12     vector<int> d (n, INF);
13     d[v] = 0;
14     for (int i=0; i<n-1; ++i)
15         for (int j=0; j<m; ++j)
16             if (d[e[j].a] < INF)
17                 d[e[j].b] = min (d[e[j].b], d[e[j].a]
18 + e[j].cost);
19 }

```

6.8 Dinic

```

1 // Description:
2 // Obtains the maximum possible flow rate given a
   network. A network is a graph with a single
   source vertex and a single sink vertex in which
   each edge has a capacity
3
4 // Problem:
5 // https://codeforces.com/gym/103708/problem/J
6
7 // Complexity:
8 //  $O(V^2 * E)$  where  $V$  is the number of vertex and  $E$ 
   is the number of edges
9
10 // Unit network
11 // A unit network is a network in which for any
   vertex except source and sink either incoming or
   outgoing edge is unique and has unit capacity (
   matching problem).
12 // Complexity on unit networks:  $O(E * \sqrt{V})$ 
13
14 // Unity capacity networks
15 // A more generic settings when all edges have unit
   capacities, but the number of incoming and
   outgoing edges is unbounded
16 // Complexity on unity capacity networks:  $O(E * \sqrt{E})$ 
17
18 // How to use:

```

```

19 // Dinic dinic = Dinic(num_vertex, source, sink);
20 // dinic.add_edge(vertex1, vertex2, capacity);
21 // cout << dinic.max_flow() << '\n';
22
23 #include <bits/stdc++.h>
24
25 #define pb push_back
26 #define mp make_pair
27 #define pii pair<int, int>
28 #define ff first
29 #define ss second
30 #define ll long long
31
32 using namespace std;
33
34 const ll INF = 1e18+10;
35
36 struct Edge {
37     int from;
38     int to;
39     ll capacity;
40     ll flow;
41     Edge* residual;
42
43     Edge() {}
44
45     Edge(int from, int to, ll capacity) : from(from),
46         to(to), capacity(capacity) {
47         flow = 0;
48     }
49
50     ll get_capacity() {
51         return capacity - flow;
52     }
53
54     ll get_flow() {
55         return flow;
56     }
57
58     void augment(ll bottleneck) {
59         flow += bottleneck;
60         residual->flow -= bottleneck;
61     }
62
63     void reverse(ll bottleneck) {
64         flow -= bottleneck;
65         residual->flow += bottleneck;
66     }
67
68     bool operator<(const Edge& e) const {
69         return true;
70     };
71
72 struct Dinic {
73     int source;
74     int sink;
75     int nodes;
76     ll flow;
77     vector<vector<Edge*>> adj;
78     vector<int> level;
79     vector<int> next;
80     vector<int> reach;
81     vector<bool> visited;
82     vector<vector<int>> path;
83
84     Dinic(int source, int sink, int nodes) : source(
85         source), sink(sink), nodes(nodes) {
86         adj.resize(nodes + 1);
87     }
88
89     void add_edge(int from, int to, ll capacity) {
90         Edge* e1 = new Edge(from, to, capacity);

```

```

90     Edge* e2 = new Edge(to, from, 0);
91     // Edge* e2 = new Edge(to, from, capacity);
92     e1->residual = e2;
93     e2->residual = e1;
94     adj[from].pb(e1);
95     adj[to].pb(e2);
96 }
97
98 bool bfs() {
99     level.assign(nodes + 1, -1);
100     queue<int> q;
101     q.push(source);
102     level[source] = 0;
103
104     while (!q.empty()) {
105         int node = q.front();
106         q.pop();
107
108         for (auto e : adj[node]) {
109             if (level[e->to] == -1 && e->
get_capacity() > 0) {
110                 level[e->to] = level[e->from] +
1;
111                 q.push(e->to);
112             }
113         }
114     }
115     return level[sink] != -1;
116 }
117
118 ll dfs(int v, ll flow) {
119     if (v == sink)
120         return flow;
121
122     int sz = adj[v].size();
123     for (int i = next[v]; i < sz; i++) {
124         Edge* e = adj[v][i];
125         if (level[e->to] == level[e->from] + 1 && e->get_capacity() > 0) {
126             ll bottleneck = dfs(e->to, min(flow,
e->get_capacity()));
127             if (bottleneck > 0) {
128                 e->augment(bottleneck);
129                 return bottleneck;
130             }
131         }
132         next[v] = i + 1;
133     }
134     return 0;
135 }
136
137 ll max_flow() {
138     flow = 0;
139     while(bfs()) {
140         next.assign(nodes + 1, 0);
141         ll sent = -1;
142         while (sent != 0) {
143             sent = dfs(source, INF);
144             flow += sent;
145         }
146     }
147     return flow;
148 }
149
150 void reachable(int v) {
151     visited[v] = true;
152
153     for (auto e : adj[v]) {
154         if (!visited[e->to] && e->get_capacity()
> 0) {
155             reach.pb(e->to);
156             visited[e->to] = true;
157             reachable(e->to);
158         }
159     }
160 }
161
162 void print_min_cut() {
163     reach.clear();
164     visited.assign(nodes + 1, false);
165     reach.pb(source);
166     reachable(source);
167
168     for (auto v : reach) {
169         for (auto e : adj[v]) {
170             if (!visited[e->to] && e->
get_capacity() == 0) {
171                 cout << e->from << ' ' << e->to
<< '\n';
172             }
173         }
174     }
175 }
176
177 ll build_path(int v, int id, ll flow) {
178     visited[v] = true;
179     if (v == sink) {
180         return flow;
181     }
182     for (auto e : adj[v]) {
183         if (!visited[e->to] && e->get_flow() > 0)
{
184             visited[e->to] = true;
185             ll bottleneck = build_path(e->to, id,
min(flow, e->get_flow()));
186             if (bottleneck > 0) {
187                 path[id].pb(e->to);
188                 e->reverse(bottleneck);
189                 return bottleneck;
190             }
191         }
192     }
193     return 0;
194 }
195
196 void print_flow_path() {
197     path.clear();
198     ll sent = -1;
199     int id = -1;
200     while (sent != 0) {
201         visited.assign(nodes + 1, false);
202         path.pb(vector<int>{});
203         sent = build_path(source, ++id, INF);
204         path[id].pb(source);
205     }
206     path.pop_back();
207
208     for (int i = 0; i < id; i++) {
209         cout << path[i].size() << '\n';
210         reverse(path[i].begin(), path[i].end());
211         for (auto e : path[i]) {
212             cout << e << ' ';
213         }
214         cout << '\n';
215     }
216 }
217
218 int main() {
219     ios::sync_with_stdio(false);
220     cin.tie(NULL);

```

```

227
228     int n, m; cin >> n >> m;
229
230     Dinic dinic = Dinic(1, n, n);
231
232     for (int i = 1; i <= m; i++) {
233         int v, u; cin >> v >> u;
234         dinic.add_edge(v, u, 1);
235     }
236
237     cout << dinic.max_flow() << '\n';
238     // dinic.print_min_cut();
239     // dinic.print_flow_path();
240
241     return 0;
242 }

```

6.9 2sat

```

1 // Description:
2 // Solves expression of the type (a v b) ^ (c v d) ^
   (e v f)
3
4 // Problem:
5 // https://cses.fi/problemset/task/1684
6
7 // Complexity:
8 // O(n + m) where n is the number of variables and m
   is the number of clauses
9
10 #include <bits/stdc++.h>
11 #define pb push_back
12 #define mp make_pair
13 #define pii pair<int, int>
14 #define ff first
15 #define ss second
16
17 using namespace std;
18
19 struct SAT {
20     int nodes;
21     int curr = 0;
22     int component = 0;
23     vector<vector<int>> adj;
24     vector<vector<int>> rev;
25     vector<vector<int>> condensed;
26     vector<pii> departure;
27     vector<bool> visited;
28     vector<int> scc;
29     vector<int> order;
30
31     // 1 to nodes
32     // nodes + 1 to 2 * nodes
33     SAT(int nodes) : nodes(nodes) {
34         adj.resize(2 * nodes + 1);
35         rev.resize(2 * nodes + 1);
36         visited.resize(2 * nodes + 1);
37         scc.resize(2 * nodes + 1);
38     }
39
40     void add_imp(int a, int b) {
41         adj[a].pb(b);
42         rev[b].pb(a);
43     }
44
45     int get_not(int a) {
46         if (a > nodes) return a - nodes;
47         return a + nodes;
48     }
49
50     void add_or(int a, int b) {
51         add_imp(get_not(a), b);
52         add_imp(get_not(b), a);

```

```

53     }
54
55     void add_nor(int a, int b) {
56         add_or(get_not(a), get_not(b));
57     }
58
59     void add_and(int a, int b) {
60         add_or(get_not(a), b);
61         add_or(a, get_not(b));
62         add_or(a, b);
63     }
64
65     void add_nand(int a, int b) {
66         add_or(get_not(a), b);
67         add_or(a, get_not(b));
68         add_or(get_not(a), get_not(b));
69     }
70
71     void add_xor(int a, int b) {
72         add_or(a, b);
73         add_or(get_not(a), get_not(b));
74     }
75
76     void add_xnor(int a, int b) {
77         add_or(get_not(a), b);
78         add_or(a, get_not(b));
79     }
80
81     void departure_time(int v) {
82         visited[v] = true;
83
84         for (auto u : adj[v]) {
85             if (!visited[u]) departure_time(u);
86         }
87
88         departure.pb(mp(++curr, v));
89     }
90
91     void find_component(int v, int component) {
92         scc[v] = component;
93         visited[v] = true;
94
95         for (auto u : rev[v]) {
96             if (!visited[u]) find_component(u,
97 component);
98         }
99     }
100
101     void topological_order(int v) {
102         visited[v] = true;
103
104         for (auto u : condensed[v]) {
105             if (!visited[u]) topological_order(u);
106         }
107
108         order.pb(v);
109     }
110
111     bool is_possible() {
112         component = 0;
113         for (int i = 1; i <= 2 * nodes; i++) {
114             if (!visited[i]) departure_time(i);
115         }
116
117         sort(departure.begin(), departure.end(),
118 greater<pii>());
119
120         visited.assign(2 * nodes + 1, false);
121
122         for (auto [_, node] : departure) {
123             if (!visited[node]) find_component(node,
124 ++component);
125         }

```

```

123     for (int i = 1; i <= nodes; i++) {
124         if (scc[i] == scc[i + nodes]) return
125         false;
126     }
127
128     return true;
129 }
130
131 int find_value(int e, vector<int> &ans) {
132     if (e > nodes && ans[e - nodes] != 2) return
133     !ans[e - nodes];
134     if (e <= nodes && ans[e + nodes] != 2) return
135     !ans[e + nodes];
136     return 0;
137 }
138
139 vector<int> find_ans() {
140     condensed.resize(component + 1);
141
142     for (int i = 1; i <= 2 * nodes; i++) {
143         for (auto u : adj[i]) {
144             if (scc[i] != scc[u]) condensed[scc[i]
145             ]].pb(scc[u]);
146         }
147
148         visited.assign(component + 1, false);
149
150         for (int i = 1; i <= component; i++) {
151             if (!visited[i]) topological_order(i);
152         }
153
154         reverse(order.begin(), order.end());
155
156         // 0 - false
157         // 1 - true
158         // 2 - no value yet
159         vector<int> ans(2 * nodes + 1, 2);
160
161         vector<vector<int>> belong(component + 1);
162
163         for (int i = 1; i <= 2 * nodes; i++) {
164             belong[scc[i]].pb(i);
165         }
166
167         for (auto p : order) {
168             for (auto e : belong[p]) {
169                 ans[e] = find_value(e, ans);
170             }
171         }
172
173         return ans;
174     }
175 };
176
177 int main() {
178     ios::sync_with_stdio(false);
179     cin.tie(NULL);
180
181     int n, m; cin >> n >> m;
182
183     SAT sat = SAT(m);
184
185     for (int i = 0; i < n; i++) {
186         char op1, op2; int a, b; cin >> op1 >> a >>
187         op2 >> b;
188         if (op1 == '+' && op2 == '+') sat.add_or(a, b
189         );
190         if (op1 == '-' && op2 == '-') sat.add_or(sat.
191         get_not(a), sat.get_not(b));
192         if (op1 == '+' && op2 == '-') sat.add_or(a,
193         sat.get_not(b));

```

```

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

```

6.10 Find Cycle

```

1 bitset<MAX> visited;
2 vector<int> path;
3 vector<int> adj[MAX];
4
5 bool dfs(int u, int p){
6
7     if (visited[u]) return false;
8
9     path.pb(u);
10    visited[u] = true;
11
12    for (auto v : adj[u]){
13        if (visited[v] and u != v and p != v){
14            path.pb(v); return true;
15        }
16
17        if (dfs(v, u)) return true;
18    }
19
20    path.pop_back();
21    return false;
22 }
23
24 bool has_cycle(int N){
25
26    visited.reset();
27
28    for (int u = 1; u <= N; ++u){
29        path.clear();
30        if (not visited[u] and dfs(u,-1))
31            return true;
32    }
33
34    return false;
35 }
36 }

```

6.11 Hld

```

1 // Description:
2 // Query path - query path (a, b) inclusive
3 // Update path - update path (a, b) inclusive
4 // Query subtree - query subtree of a
5 // Update subtree - update subtree of a
6 // Update - update vertex or edge
7 // Lca - get lowest common ancestor of a and b
8 // Search - perform a binary search to find the last
9 // on the path from a to the root
10
11 // Problem:
12 // https://codeforces.com/gym/101908/problem/L
13
14 // Complexity:

```



```

15 // O(log ^2 n) for both query and update
16
17 // How to use:
18 // HLD hld = HLD(n + 1, adj)
19
20 // Notes
21 // Change the root of the tree on the constructor if
    it's different from 1
22 // Use together with Segtree
23 // For edge query and update, update the child node
    with the value of the edge
24 // edge(a, b) and parent[a] = b    => update vertex a
25
26 struct HLD {
27     vector<int> parent;
28     vector<int> pos;
29     vector<int> head;
30     vector<int> subtree_size;
31     vector<int> level;
32     vector<int> heavy_child;
33     vector<ftype> subtree_weight;
34     vector<ftype> path_weight;
35     vector<vector<int>>> adj;
36     vector<int> at;
37     Segtree seg = Segtree(0);
38     int cpos;
39     int n;
40     int root;
41
42     HLD() {}
43
44     HLD(int n, vector<vector<int>>& adj, int root = 1)
        : adj(adj), n(n), root(root) {
45         seg = Segtree(n);
46         cpos = 0;
47         at.resize(n);
48         parent.resize(n);
49         pos.resize(n);
50         head.resize(n);
51         subtree_size.assign(n, 1);
52         level.assign(n, 0);
53         heavy_child.assign(n, -1);
54         parent[root] = -1;
55         dfs(root, -1);
56         decompose(root, -1);
57     }
58
59     void dfs(int v, int p) {
60         parent[v] = p;
61         if (p != -1) level[v] = level[p] + 1;
62         for (auto u : adj[v]) {
63             if (u != p) {
64                 dfs(u, v);
65                 subtree_size[v] += subtree_size[u];
66                 if (heavy_child[v] == -1 || subtree_size[u] >
                    subtree_size[heavy_child[v]]) heavy_child[v] = u;
67             }
68         }
69     }
70
71     void decompose(int v, int chad) {
72         // start a new path
73         if (chad == -1) chad = v;
74
75         // consecutive ids in the hld path
76         at[cpos] = v;
77         pos[v] = cpos++;
78         head[v] = chad;
79
80         // if not a leaf
81         if (heavy_child[v] != -1) decompose(heavy_child[v],
            chad);
82
83         // light child
84         for (auto u : adj[v]){
85             // start new path
86             if (u != parent[v] && u != heavy_child[v])
                decompose(u, -1);
87         }
88     }
89
90     ll query_path(int a, int b) {
91         if(pos[a] < pos[b]) swap(a, b);
92
93         if(head[a] == head[b]) return seg.query(pos[b],
            pos[a]);
94         return seg.f(seg.query(pos[head[a]], pos[a]),
            query_path(parent[head[a]], b));
95     }
96
97     ftype query_subtree(int a) {
98         return seg.query(pos[a], pos[a] + subtree_size[a]
            - 1);
99     }
100
101     void update_path(int a, int b, int x) {
102         if(pos[a] < pos[b]) swap(a, b);
103
104         if(head[a] == head[b]) return (void)seg.update(
            pos[b], pos[a], x);
105         seg.update(pos[head[a]], pos[a], x); update_path(
            parent[head[a]], b, x);
106     }
107
108     void update_subtree(int a, int val) {
109         seg.update(pos[a], pos[a] + subtree_size[a] - 1,
            val);
110     }
111
112     void update(int a, int val) {
113         seg.update(pos[a], pos[a], val);
114     }
115
116     //edge
117     void update(int a, int b, int val) {
118         if (level[a] > level[b]) swap(a, b);
119         update(b, val);
120     }
121
122     int lca(int a, int b) {
123         if(pos[a] < pos[b]) swap(a, b);
124         return head[a] == head[b] ? b : lca(parent[head[a]
            ], b);
125     }
126
127     void search(int a) {
128         a = parent[a];
129         if (a == -1) return;
130         if (seg.query(pos[head[a]], pos[head[a]]+
            subtree_size[head[a]]-1) + pos[a]-pos[head[a]]+1
            == subtree_size[head[a]]) {
131             seg.update(pos[head[a]], pos[a], 1);
132             return search(parent[head[a]]);
133         }
134         int l = pos[head[a]], r = pos[a]+1;
135         while (l < r) {
136             int m = (l+r)/2;
137             if (seg.query(m, m+subtree_size[at[m]]-1) + pos
                [a]-m+1 == subtree_size[at[m]]) {
138                 r = m;
139             }
140             else l = m+1;
141         }
142         seg.update(l, pos[a], 1);
143     }

```

```
144 };
```

6.12 Cycle Path Recovery

```
1 int n;
2 vector<vector<int>> adj;
3 vector<char> color;
4 vector<int> parent;
5 int cycle_start, cycle_end;
6
7 bool dfs(int v) {
8     color[v] = 1;
9     for (int u : adj[v]) {
10         if (color[u] == 0) {
11             parent[u] = v;
12             if (dfs(u))
13                 return true;
14         } else if (color[u] == 1) {
15             cycle_end = v;
16             cycle_start = u;
17             return true;
18         }
19     }
20     color[v] = 2;
21     return false;
22 }
23
24 void find_cycle() {
25     color.assign(n, 0);
26     parent.assign(n, -1);
27     cycle_start = -1;
28
29     for (int v = 0; v < n; v++) {
30         if (color[v] == 0 && dfs(v))
31             break;
32     }
33
34     if (cycle_start == -1) {
35         cout << "Acyclic" << endl;
36     } else {
37         vector<int> cycle;
38         cycle.push_back(cycle_start);
39         for (int v = cycle_end; v != cycle_start; v =
40             parent[v])
41             cycle.push_back(v);
42         cycle.push_back(cycle_start);
43         reverse(cycle.begin(), cycle.end());
44
45         cout << "Cycle found: ";
46         for (int v : cycle)
47             cout << v << " ";
48         cout << endl;
49     }
50 }
```

6.13 Centroid Decomposition

```
1 int n;
2 vector<set<int>> adj;
3 vector<char> ans;
4
5 vector<bool> removed;
6
7 vector<int> subtree_size;
8
9 int dfs(int u, int p = 0) {
10     subtree_size[u] = 1;
11
12     for(int v : adj[u]) {
13         if(v != p && !removed[v]) {
14             subtree_size[u] += dfs(v, u);
15         }
16     }
```

```
16     }
17
18     return subtree_size[u];
19 }
20
21 int get_centroid(int u, int sz, int p = 0) {
22     for(int v : adj[u]) {
23         if(v != p && !removed[v]) {
24             if(subtree_size[v]*2 > sz) {
25                 return get_centroid(v, sz, u);
26             }
27         }
28     }
29
30     return u;
31 }
32
33 char get_next(char c) {
34     if (c != 'Z') return c + 1;
35     return '$';
36 }
37
38 bool flag = true;
39
40 void solve(int node, char c) {
41     int center = get_centroid(node, dfs(node));
42     ans[center] = c;
43     removed[center] = true;
44
45     for (auto u : adj[center]) {
46         if (!removed[u]) {
47             char next = get_next(c);
48             if (next == '$') {
49                 flag = false;
50                 return;
51             }
52             solve(u, next);
53         }
54     }
55 }
56
57 int32_t main(){
58     ios::sync_with_stdio(false);
59     cin.tie(NULL);
60
61     cin >> n;
62     adj.resize(n + 1);
63     ans.resize(n + 1);
64     removed.resize(n + 1);
65     subtree_size.resize(n + 1);
66
67     for (int i = 1; i <= n - 1; i++) {
68         int u, v; cin >> u >> v;
69         adj[u].insert(v);
70         adj[v].insert(u);
71     }
72
73     solve(1, 'A');
74
75     if (!flag) cout << "Impossible!\n";
76     else {
77         for (int i = 1; i <= n; i++) {
78             cout << ans[i] << ' ';
79         }
80         cout << '\n';
81     }
82
83     return 0;
84 }
```

6.14 Tarjan Bridge

```
1 // Description:
```

```

2 // Find a bridge in a connected undirected graph
3 // A bridge is an edge so that if you remove that
  edge the graph is no longer connected
4
5 // Problem:
6 // https://cses.fi/problemset/task/2177/
7
8 // Complexity:
9 //  $O(V + E)$  where  $V$  is the number of vertices and  $E$ 
  is the number of edges
10
11 int n;
12 vector<vector<int>> adj;
13
14 vector<bool> visited;
15 vector<int> tin, low;
16 int timer;
17
18 void dfs(int v, int p) {
19     visited[v] = true;
20     tin[v] = low[v] = timer++;
21     for (int to : adj[v]) {
22         if (to == p) continue;
23         if (visited[to]) {
24             low[v] = min(low[v], tin[to]);
25         } else {
26             dfs(to, v);
27             low[v] = min(low[v], low[to]);
28             if (low[to] > tin[v]) {
29                 IS_BRIDGE(v, to);
30             }
31         }
32     }
33 }
34
35 void find_bridges() {
36     timer = 0;
37     visited.assign(n, false);
38     tin.assign(n, -1);
39     low.assign(n, -1);
40     for (int i = 0; i < n; ++i) {
41         if (!visited[i])
42             dfs(i, -1);
43     }
44 }

```

6.15 Small To Large

```

1 // Problem:
2 // https://codeforces.com/contest/600/problem/E
3
4 void process_colors(int curr, int parent) {
5
6     for (int n : adj[curr]) {
7         if (n != parent) {
8             process_colors(n, curr);
9
10             if (colors[curr].size() < colors[n].size
11                 ()) {
12                 sum_num[curr] = sum_num[n];
13                 vmax[curr] = vmax[n];
14                 swap(colors[curr], colors[n]);
15             }
16             for (auto [item, vzs] : colors[n]) {
17                 if (colors[curr][item] + vzs > vmax[curr]
18                     ) {
19                     vmax[curr] = colors[curr][item] +
20                         vzs;
21                     sum_num[curr] = item;
22                 } else if (colors[curr][item] + vzs ==
23                     ) {
24                     sum_num[curr] += item;
25                 }
26             }
27         }
28     }
29 }

```

```

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

```

6.16 Tree Diameter

```

1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX = 3e5+17;
6
7 vector<int> adj[MAX];
8 bool visited[MAX];
9
10 int max_depth = 0, max_node = 1;
11
12 void dfs (int v, int depth) {
13     visited[v] = true;
14
15     if (depth > max_depth) {
16         max_depth = depth;
17         max_node = v;
18     }
19
20     for (auto u : adj[v]) {
21         if (!visited[u]) dfs(u, depth + 1);
22     }
23 }
24
25 int tree_diameter() {
26     dfs(1, 0);
27     max_depth = 0;
28     for (int i = 0; i < MAX; i++) visited[i] = false;
29     dfs(max_node, 0);
30     return max_depth;
31 }

```

6.17 Dijkstra

```
1 const int MAX = 2e5+7;
2 const int INF = 1000000000;
3 vector<vector<pair<int, int>>> adj(MAX);
4
5 void dijkstra(int s, vector<int> & d, vector<int> & p
6 ) {
7     int n = adj.size();
8     d.assign(n, INF);
9     p.assign(n, -1);
10
11     d[s] = 0;
12     set<pair<int, int>> q;
13     q.insert({0, s});
14     while (!q.empty()) {
15         int v = q.begin()->second;
16         q.erase(q.begin());
17
18         for (auto edge : adj[v]) {
19             int to = edge.first;
20             int len = edge.second;
21
22             if (d[v] + len < d[to]) {
23                 q.erase({d[to], to});
24                 d[to] = d[v] + len;
25                 p[to] = v;
26                 q.insert({d[to], to});
27             }
28         }
29     }
30
31     vector<int> restore_path(int s, int t) {
32         vector<int> path;
33
34         for (int v = t; v != s; v = p[v])
35             path.push_back(v);
36         path.push_back(s);
37
38         reverse(path.begin(), path.end());
39         return path;
40     }
41
42     int adj[MAX][MAX];
43     int dist[MAX];
44     int minDistance(int dist[], bool sptSet[], int V) {
45         int min = INT_MAX, min_index;
46
47         for (int v = 0; v < V; v++)
48             if (sptSet[v] == false && dist[v] <= min)
49                 min = dist[v], min_index = v;
50
51         return min_index;
52     }
53
54     void dijkstra(int src, int V) {
55         bool sptSet[V];
56         for (int i = 0; i < V; i++)
57             dist[i] = INT_MAX, sptSet[i] = false;
58
59         dist[src] = 0;
60
61         for (int count = 0; count < V - 1; count++) {
62             int u = minDistance(dist, sptSet, V);
63
64             sptSet[u] = true;
65
66             for (int v = 0; v < V; v++)
67                 if (!sptSet[v] && adj[u][v]
68                     && dist[u] != INT_MAX
```

```
71         && dist[u] + adj[u][v] < dist[v])
72             dist[v] = dist[u] + adj[u][v];
73     }
74 }
```

6.18 Kruskall

```
1 struct DSU {
2     int n;
3     vector<int> link, sizes;
4
5     DSU(int n) {
6         this->n = n;
7         link.assign(n+1, 0);
8         sizes.assign(n+1, 1);
9
10        for (int i = 0; i <= n; i++)
11            link[i] = i;
12    }
13
14    int find(int x) {
15        while (x != link[x])
16            x = link[x];
17
18        return x;
19    }
20
21    bool same(int a, int b) {
22        return find(a) == find(b);
23    }
24
25    void unite(int a, int b) {
26        a = find(a);
27        b = find(b);
28
29        if (a == b) return;
30
31        if (sizes[a] < sizes[b])
32            swap(a, b);
33
34        sizes[a] += sizes[b];
35        link[b] = a;
36    }
37 };
38
39 struct Edge {
40     int u, v;
41     long long weight;
42
43     Edge() {}
44
45     Edge(int u, int v, long long weight) : u(u), v(v), weight(weight) {}
46
47     bool operator<(const Edge& other) const {
48         return weight < other.weight;
49     }
50
51     bool operator>(const Edge& other) const {
52         return weight > other.weight;
53     }
54 };
55
56 vector<Edge> kruskal(vector<Edge> edges, int n) {
57     vector<Edge> result; // arestas da MST
58     long long cost = 0;
59
60     sort(edges.begin(), edges.end());
61
62     DSU dsu(n);
63
64     for (auto e : edges) {
65         if (!dsu.same(e.u, e.v)) {
```

```

66         cost += e.weight;
67         result.push_back(e);
68         dsu.unite(e.u, e.v);
69     }
70 }
71
72 return result;
73 }

```

7 Geometry

7.1 2d

```

1 #define vp vector<point>
2 #define ld long double
3 const ld EPS = 1e-6;
4 const ld PI = acos(-1);
5
6 // typedef ll cod;
7 // bool eq(cod a, cod b){ return (a==b); }
8 typedef ld cod;
9 bool eq(cod a, cod b){ return abs(a - b) <= EPS; }
10
11 struct point{
12     cod x, y;
13     int id;
14     point(cod x=0, cod y=0): x(x), y(y){}
15
16     point operator+(const point &o) const{ return {x+
17     o.x, y+o.y}; }
18     point operator-(const point &o) const{ return {x-
19     o.x, y-o.y}; }
20     point operator*(cod t) const{ return {x*t, y*t};
21     }
22     point operator/(cod t) const{ return {x/t, y/t};
23     }
24     cod operator*(const point &o) const{ return x * o
25     .x + y * o.y; }
26     cod operator^(const point &o) const{ return x * o
27     .y - y * o.x; }
28     bool operator<(const point &o) const{
29         return (eq(x, o.x) ? y < o.y : x < o.x);
30     }
31     bool operator==(const point &o) const{
32         return eq(x, o.x) and eq(y, o.y);
33     }
34     friend ostream& operator<<(ostream& os, point p) {
35         return os << "(" << p.x << "," << p.y << ")"; }
36 }
37
38 int ccw(point a, point b, point e){ // -1=dir; 0=
39     collinear; 1=esq;
40     cod tmp = (b-a) ^ (e-a); // vector from a to b
41     return (tmp > EPS) - (tmp < -EPS);
42 }
43
44 ld norm(point a){ // Modulo
45     return sqrt(a * a);
46 }
47
48 cod norm2(point a){
49     return a * a;
50 }
51
52 bool nulo(point a){
53     return (eq(a.x, 0) and eq(a.y, 0));
54 }
55
56 point rotccw(point p, ld a){
57     // a = PI*a/180; // graus
58     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+
59     p.x*sin(a)));
60 }
61
62 point rot90cw(point a) { return point(a.y, -a.x); }
63
64 point rot90ccw(point a) { return point(-a.y, a.x); }

```

```

52
53 ld proj(point a, point b){ // a sobre b
54     return a*b/norm(b);
55 }
56
57 ld angle(point a, point b){ // em radianos
58     ld ang = a*b / norm(a) / norm(b);
59     return acos(max(min(ang, (ld)1), (ld)-1));
60 }
61
62 ld angle_vec(point v){
63     // return 180/PI*atan2(v.x, v.y); // graus
64     return atan2(v.x, v.y);
65 }
66
67 ld order_angle(point a, point b){ // from a to b ccw
68     (a in front of b)
69     ld aux = angle(a,b)*180/PI;
70     return ((a^b)<=0 ? aux:360-aux);
71 }
72
73 bool angle_less(point a1, point b1, point a2, point
74     b2){ // ang(a1,b1) <= ang(a2,b2)
75     point p1((a1*b1), abs((a1^b1)));
76     point p2((a2*b2), abs((a2^b2)));
77     return (p1^p2) <= 0;
78 }
79
80 ld area(vp &p){ // (points sorted)
81     ld ret = 0;
82     for(int i=2;i<(int)p.size();i++)
83         ret += (p[i]-p[0])^(p[i-1]-p[0]);
84     return abs(ret/2);
85 }
86
87 ld areaT(point &a, point &b, point &c){
88     return abs((b-a)^(c-a))/2.0;
89 }
90
91 point center(vp &A){
92     point c = point();
93     int len = A.size();
94     for(int i=0;i<len;i++)
95         c=c+A[i];
96     return c/len;
97 }
98
99 point forca_mod(point p, ld m){
100     ld cm = norm(p);
101     if(cm<EPS) return point();
102     return point(p.x*m/cm,p.y*m/cm);
103 }
104
105 ld param(point a, point b, point v){
106     // v = t*(b-a) + a // return t;
107     // assert(line(a, b).inside_seg(v));
108     return ((v-a) * (b-a)) / ((b-a) * (b-a));
109 }
110
111 bool simetric(vp &a){ //ordered
112     int n = a.size();
113     point c = center(a);
114     if(n&1) return false;
115     for(int i=0;i<n/2;i++)
116         if(ccw(a[i], a[i+n/2], c) != 0)
117             return false;
118     return true;
119 }
120
121 point mirror(point m1, point m2, point p){
122     // mirror point p around segment m1m2
123     point seg = m2-m1;
124     ld t0 = ((p-m1)*seg) / (seg*seg);
125     point ort = m1 + seg*t0;
126     point pm = ort-(p-ort);
127     return pm;

```

```

123
124 ///////////////
125 // Line //
126 ///////////////
127
128 struct line{
129     point p1, p2;
130     cod a, b, c; // ax+by+c = 0;
131     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
132     line(point p1=0, point p2=0): p1(p1), p2(p2){
133         a = p1.y - p2.y;
134         b = p2.x - p1.x;
135         c = p1 ^ p2;
136     }
137     line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c){
138         // Gera os pontos p1 p2 dados os coeficientes
139         // isso aqui eh um lixo mas quebra um galho
140         kkkkkk
141         if(b==0){
142             p1 = point(1, -c/a);
143             p2 = point(0, -c/a);
144         }else{
145             p1 = point(1, (-c-a*1)/b);
146             p2 = point(0, -c/b);
147         }
148
149         cod eval(point p){
150             return a*p.x+b*p.y+c;
151         }
152         bool inside(point p){
153             return eq(eval(p), 0);
154         }
155         point normal(){
156             return point(a, b);
157         }
158
159         bool inside_seg(point p){
160             return (
161                 ((p1-p) ^ (p2-p)) == 0 and
162                 ((p1-p) * (p2-p)) <= 0
163             );
164         }
165     };
166 };
167
168 // be careful with precision error
169 vp inter_line(line l1, line l2){
170     ld det = l1.a*l2.b - l1.b*l2.a;
171     if(det==0) return {};
172     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
173     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
174     return {point(x, y)};
175 }
176
177 // segments not collinear
178 vp inter_seg(line l1, line l2){
179     vp ans = inter_line(l1, l2);
180     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.inside_seg(ans[0]))
181         return {};
182     return ans;
183 }
184 bool seg_has_inter(line l1, line l2){
185     return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.p2, l2.p2) < 0 and
186         ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.p2, l1.p2) < 0;
187 }
188
189 ld dist_seg(point p, point a, point b){ // point -
190     seg
191     if((p-a)*(b-a) < EPS) return norm(p-a);
192     if((p-b)*(a-b) < EPS) return norm(p-b);
193     return abs((p-a)^(b-a)) / norm(b-a);
194 }
195
196 ld dist_line(point p, line l){ // point - line
197     return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
198 }
199
200 line bisector(point a, point b){
201     point d = (b-a)*2;
202     return line(d.x, d.y, a*a - b*b);
203 }
204
205 line perpendicular(line l, point p){ // passes
206     through p
207     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
208 }
209
210 // Circle //
211 ///////////////
212
213 struct circle{
214     point c; cod r;
215     circle() : c(0, 0), r(0){}
216     circle(const point o) : c(o), r(0){}
217     circle(const point a, const point b){
218         c = (a+b)/2;
219         r = norm(a-c);
220     }
221     circle(const point a, const point b, const point
222             cc){
223         assert(ccw(a, b, cc) != 0);
224         c = inter_line(bisector(a, b), bisector(b, cc
225             ))[0];
226         r = norm(a-c);
227     }
228     bool inside(const point &a) const{
229         return norm(a - c) <= r + EPS;
230     }
231 };
232
233 pair<point, point> tangent_points(circle cr, point p)
234 {
235     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
236     point p1 = rotccw(cr.c-p, -theta);
237     point p2 = rotccw(cr.c-p, theta);
238     assert(d1 >= cr.r);
239     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
240     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
241     return {p1, p2};
242 }
243
244 circle incircle(point p1, point p2, point p3){
245     ld m1 = norm(p2-p3);
246     ld m2 = norm(p1-p3);
247     ld m3 = norm(p1-p2);
248     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
249     ld s = 0.5*(m1+m2+m3);
250     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
251     return circle(c, r);
252 }
253
254 circle circumcircle(point a, point b, point c) {
255     circle ans;
256     point u = point((b-a).y, -(b-a).x);
257     point v = point((c-a).y, -(c-a).x);
258     point n = (c-b)*0.5;
259     ld t = (u^n)/(v^u);
260     ans.c = ((a+c)*0.5) + (v*t);

```

```

259     ans.r = norm(ans.c-a);
260     return ans;
261 }
262
263 vp inter_circle_line(circle C, line L){
264     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.
265     p1)*(ab) / (ab*ab));
266     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
267     / (ab*ab);
268     if (h2 < -EPS) return {};
269     if (eq(h2, 0)) return {p};
270     point h = (ab/norm(ab)) * sqrt(h2);
271     return {p - h, p + h};
272 }
273
274 vp inter_circle(circle C1, circle C2){
275     if(C1.c == C2.c) { assert(C1.r != C2.r); return
276     {}; }
277     point vec = C2.c - C1.c;
278     ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
279     ;
280     ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
281     C1.r*C1.r - p*p*d2;
282     if (sum*sum < d2 or dif*dif > d2) return {};
283     point mid = C1.c + vec*p, per = point(-vec.y, vec
284     .x) * sqrt(max((ld)0, h2) / d2);
285     if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
286     return {mid + per, mid - per};
287 }
288
289 // minimum circle cover O(n) amortizado
290 circle min_circle_cover(vp v){
291     random_shuffle(v.begin(), v.end());
292     circle ans;
293     int n = v.size();
294     for(int i=0;i<n;i++) if(!ans.inside(v[i])){
295         ans = circle(v[i]);
296         for(int j=0;j<i;j++) if(!ans.inside(v[j])){
297             ans = circle(v[i], v[j]);
298             for(int k=0;k<j;k++) if(!ans.inside(v[k]))
299                 ans = circle(v[i], v[j], v[k]);
300         }
301     }
302     return ans;
303 }

```

8 Algorithms

8.1 Lis

```

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

```

8.2 Delta-encoding

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int n, q;
6     cin >> n >> q;
7     int [n];
8     int delta[n+2];
9
10    while(q--){
11        int l, r, x;
12        cin >> l >> r >> x;
13        delta[l] += x;
14        delta[r+1] -= x;
15    }
16
17    int curr = 0;
18    for(int i=0; i < n; i++){
19        curr += delta[i];
20        v[i] = curr;
21    }
22
23    for(int i=0; i < n; i++){
24        cout << v[i] << ' ';
25    }
26    cout << '\n';
27
28    return 0;
29 }

```

8.3 Subsets

```

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

```

8.4 Binary Search Last True

```

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

```

8.5 Ternary Search

```

1 double ternary_search(double l, double r) {
2     double eps = 1e-9; //set the error
3     limit here
4     while (r - l > eps) {
5         double m1 = l + (r - l) / 3;
6         double m2 = r - (r - l) / 3;
7         double f1 = f(m1); //evaluates the
8         function at m1
9     }
10 }

```

```

7         double f2 = f(m2);           //evaluates the
function at m2
8         if (f1 < f2)
9             l = m1;
10        else
11            r = m2;
12    }
13    return f(l);                       //return the
maximum of f(x) in [l, r]
14 }

```

8.6 Binary Search First True

```

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

```

8.7 Biggest K

```

1 // Description: Gets sum of k biggest or k smallest
elements in an array
2
3 // Problem: https://atcoder.jp/contests/abc306/tasks/
abc306_e
4
5 // Complexity: O(log n)
6
7 struct SetSum {
8     ll s = 0;
9     multiset<ll> mt;
10    void add(ll x){
11        mt.insert(x);
12        s += x;
13    }
14    int pop(ll x){
15        auto f = mt.find(x);
16        if(f == mt.end()) return 0;
17        mt.erase(f);
18        s -= x;
19        return 1;
20    }
21 };
22
23 struct BigK {
24     int k;
25     SetSum gt, mt;
26     BigK(int _k){
27         k = _k;
28     }
29     void balancear(){
30         while((int)gt.mt.size() < k && (int)mt.mt.
size()){
31             auto p = (prev(mt.mt.end()));
32             gt.add(*p);
33             mt.pop(*p);
34         }
35         while((int)mt.mt.size() && (int)gt.mt.size()
&&
36             *(gt.mt.begin()) < *(prev(mt.mt.end())) ){
37             ll u = *(gt.mt.begin());
38             ll v = *(prev(mt.mt.end()));
39             gt.pop(u); mt.pop(v);

```

```

40             gt.add(v); mt.add(u);
41         }
42     }
43     void add(ll x){
44         mt.add(x);
45         balancear();
46     }
47     void rem(ll x){
48         //x = -x;
49         if(mt.pop(x) == 0)
50             gt.pop(x);
51         balancear();
52     }
53 };
54
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(NULL);
58
59     int n, k, q; cin >> n >> k >> q;
60
61     BigK big = BigK(k);
62
63     int arr[n] = {};
64
65     while (q--) {
66         int pos, num; cin >> pos >> num;
67         pos--;
68         big.rem(arr[pos]);
69         arr[pos] = num;
70         big.add(arr[pos]);
71
72         cout << big.gt.s << '\n';
73     }
74
75     return 0;
76 }

```

9 Data Structures

9.1 Ordered Set

```

1 // Description:
2 // insert(k) - add element k to the ordered set
3 // erase(k) - remove element k from the ordered set
4 // erase(it) - remove element it points to from the
ordered set
5 // order_of_key(k) - returns number of elements
strictly smaller than k
6 // find_by_order(n) - return an iterator pointing to
the k-th element in the ordered set (counting
from zero).
7
8 // Problem:
9 // https://cses.fi/problemset/task/2169/
10
11 // Complexity:
12 // O(log n) for all operations
13
14 // How to use:
15 // ordered_set<int> os;
16 // cout << os.order_of_key(1) << '\n';
17 // cout << os.find_by_order(1) << '\n';
18
19 // Notes
20 // The ordered set only contains different elements
21 // By using less_equal<T> instead of less<T> on using
ordered_set declaration
22 // The ordered_set becomes an ordered_multiset
23 // So the set can contain elements that are equal
24
25 #include <ext/pb_ds/assoc_container.hpp>

```



```

26 #include <ext/pb_ds/tree_policy.hpp>
27
28 using namespace __gnu_pbds;
29 template <typename T>
30 using ordered_set = tree<T,null_type,less<T>,
    rb_tree_tag,tree_order_statistics_node_update>;
31
32 void Erase(ordered_set<int>& a, int x){
33     int r = a.order_of_key(x);
34     auto it = a.find_by_order(r);
35     a.erase(it);
36 }

```

9.2 Priority Queue

```

1 // Description:
2 // Keeps the largest (by default) element at the top
  of the queue
3
4 // Problem:
5 // https://cses.fi/problemset/task/1164/
6
7 // Complexity:
8 // O(log n) for push and pop
9 // O(1) for looking at the element at the top
10
11 // How to use:
12 // priority_queue<int> pq;
13 // pq.push(1);
14 // pq.top();
15 // pq.pop()
16
17 // Notes
18 // To use the priority queue keeping the smallest
  element at the top
19
20 priority_queue<int, vector<int>, greater<int>> pq;

```

9.3 Dsu

```

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

```

```

33         b = find(b);
34
35         if (a == b) return;
36
37         if (sizes[a] < sizes[b])
38             swap(a, b);
39
40         sizes[a] += sizes[b];
41         link[b] = a;
42     }
43
44     int size(int x) {
45         return sizes[x];
46     }
47 };
48
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(NULL);
52
53     int cities, roads; cin >> cities >> roads;
54     vector<int> final_roads;
55     int ans = 0;
56     DSU dsu = DSU(cities);
57     for (int i = 0, a, b; i < roads; i++) {
58         cin >> a >> b;
59         dsu.unite(a, b);
60     }
61
62     for (int i = 2; i <= cities; i++) {
63         if (!dsu.same(1, i)) {
64             ans++;
65             final_roads.push_back(i);
66             dsu.unite(1, i);
67         }
68     }
69
70     cout << ans << '\n';
71     for (auto e : final_roads) {
72         cout << "1 " << e << '\n';
73     }
74
75 }

```

9.4 Two Sets

```

1 // Description
2 // The values are divided in two multisets so that
  one of them contain all values that are
3 // smaller than the median and the other one contains
  all values that are greater or equal to the
  median.
4
5 // Problem:
6 // https://atcoder.jp/contests/abc306/tasks/abc306_e
7 // Problem I - Maratona Feminina de çãProgramao da
  Unicamp 2023
8 // https://codeforces.com/group/WYIydkIPyE/contest
  /450037/attachments
9
10 // Complexity:
11 // Add and remove elements - O(log n)
12 // Return sum of biggest or smallest set or return
  the median - O(1)
13
14 using ll = long long;
15
16 struct TwoSets {
17     multiset<int> small;
18     multiset<int> big;
19     ll sums = 0;
20     ll sumb = 0;
21     int n = 0;

```

```

22
23 int size_small() {
24     return small.size();
25 }
26
27 int size_big() {
28     return big.size();
29 }
30
31 void balance() {
32     while (size_small() > n / 2) {
33         int v = *small.rbegin();
34         small.erase(prev(small.end()));
35         big.insert(v);
36         sums -= v;
37         sumb += v;
38     }
39     while (size_big() > n - n / 2) {
40         int v = *big.begin();
41         big.erase(big.begin());
42         small.insert(v);
43         sumb -= v;
44         sums += v;
45     }
46 }
47
48 void add(int x) {
49     n++;
50     small.insert(x);
51     sums += x;
52     while (!small.empty() && *small.rbegin() > *big.
53         begin()) {
54         int v = *small.rbegin();
55         small.erase(prev(small.end()));
56         big.insert(v);
57         sums -= v;
58         sumb += v;
59     }
60     balance();
61 }
62
63 bool rem(int x) {
64     n--;
65     auto it1 = small.find(x);
66     auto it2 = big.find(x);
67     bool flag = false;
68     if (it1 != small.end()) {
69         sums -= *it1;
70         small.erase(it1);
71         flag = true;
72     } else if (it2 != big.end()) {
73         sumb -= *it2;
74         big.erase(it2);
75         flag = true;
76     }
77     balance();
78     return flag;
79 }
80
81 ll sum_small() {
82     return sums;
83 }
84
85 ll sum_big() {
86     return sumb;
87 }
88
89 int median() {
90     return *big.begin();
91 };

```

9.5 Dynamic Implicit Sparse

```

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

```

```

63         seg[pos] = val;
64
65     return;
66 }
67
68 int m = (ini + fim) >> 1;
69
70 if (id <= m) {
71     if (e[pos] == 0) e[pos] = create();
72     update(e[pos], ini, m, id, val);
73 } else {
74     if (d[pos] == 0) d[pos] = create();
75     update(d[pos], m + 1, fim, id, val);
76 }
77
78 seg[pos] = f(seg[e[pos]], seg[d[pos]]);
79 }
80
81 ftype query(int p, int q) {
82     return query(1, 1, n, p, q);
83 }
84
85 void update(int id, int val) {
86     update(1, 1, n, id, val);
87 }
88 };

```

9.6 Segtree2d

```

1 // Description:
2 // Indexed at zero
3 // Given a N x M grid, where i represents the row and
4 // j the column, perform the following operations
5 // update(j, i) - update the value of grid[i][j]
6 // query(j1, j2, i1, i2) - return the sum of values
7 // inside the rectangle
8 // defined by grid[i1][j1] and grid[i2][j2] inclusive
9 // Problem:
10 // https://cses.fi/problemset/task/1739/
11 // Complexity:
12 // Time complexity:
13 // O(log N * log M) for both query and update
14 // O(N * M) for build
15 // Memory complexity:
16 // 4 * M * N
17
18 // How to use:
19 // Segtree2D seg = Segtree2D(n, n);
20 // vector<vector<int>> v(n, vector<int>(n));
21 // seg.build(v);
22
23 // Notes
24 // Indexed at zero
25
26 struct Segtree2D {
27     const int MAXN = 1025;
28     int N, M;
29
30     vector<vector<int>> seg;
31
32     Segtree2D(int N, int M) {
33         this->N = N;
34         this->M = M;
35         seg.resize(2*MAXN, vector<int>(2*MAXN));
36     }
37
38     void buildY(int noX, int lX, int rX, int noY, int
39     lY, int rY, vector<vector<int>> &v){
40         if(lY == rY){
41             if(lX == rX){
42                 seg[noX][noY] = v[rX][rY];

```

```

42             }else{
43                 seg[noX][noY] = seg[2*noX+1][noY] +
44                 seg[2*noX+2][noY];
45             }
46         }else{
47             int m = (lY+rY)/2;
48
49             buildY(noX, lX, rX, 2*noY+1, lY, m, v);
50             buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);
51
52             seg[noX][noY] = seg[noX][2*noY+1] + seg[
53             noX][2*noY+2];
54         }
55     }
56
57     void buildX(int noX, int lX, int rX, vector<
58     vector<int>> &v){
59         if(lX != rX){
60             int m = (lX+rX)/2;
61
62             buildX(2*noX+1, lX, m, v);
63             buildX(2*noX+2, m+1, rX, v);
64         }
65
66         buildY(noX, lX, rX, 0, 0, M - 1, v);
67     }
68
69     void updateY(int noX, int lX, int rX, int noY,
70     int lY, int rY, int y){
71         if(lY == rY){
72             if(lX == rX){
73                 seg[noX][noY] = !seg[noX][noY];
74             }else{
75                 seg[noX][noY] = seg[2*noX+1][noY] +
76                 seg[2*noX+2][noY];
77             }
78         }else{
79             int m = (lY+rY)/2;
80
81             if(y <= m){
82                 updateY(noX, lX, rX, 2*noY+1, lY, m, y
83                 );
84             }else if(m < y){
85                 updateY(noX, lX, rX, 2*noY+2, m+1, rY
86                 , y);
87             }
88
89             seg[noX][noY] = seg[noX][2*noY+1] + seg[
90             noX][2*noY+2];
91         }
92     }
93
94     void updateX(int noX, int lX, int rX, int x, int
95     y){
96         int m = (lX+rX)/2;
97
98         if(lX != rX){
99             if(x <= m){
100                 updateX(2*noX+1, lX, m, x, y);
101             }else if(m < x){
102                 updateX(2*noX+2, m+1, rX, x, y);
103             }
104         }
105
106         updateY(noX, lX, rX, 0, 0, M - 1, y);
107     }
108
109     int queryY(int noX, int noY, int lY, int rY, int
110     aY, int bY){
111         if(aY <= lY && rY <= bY) return seg[noX][noY
112         ];
113
114         int m = (lY+rY)/2;

```

```

104         if(bY <= m) return queryY(noX, 2*noY+1, lY, m, aY, bY);
105     , aY, bY);
106     if(m < aY) return queryY(noX, 2*noY+2, m+1, rY, aY, bY);
107     rY, aY, bY);
108     return queryY(noX, 2*noY+1, lY, m, aY, bY) + queryY(noX, 2*noY+2, m+1, rY, aY, bY);
109 }
110
111 int queryX(int noX, int lX, int rX, int aX, int bX, int aY, int bY){
112     if(aX <= lX && rX <= bX) return queryY(noX, 0, 0, M - 1, aY, bY);
113
114     int m = (lX+rX)/2;
115
116     if(bX <= m) return queryX(2*noX+1, lX, m, aX, bX, aY, bY);
117     if(m < aX) return queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
118
119     return queryX(2*noX+1, lX, m, aX, bX, aY, bY) + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
120 }
121
122 void build(vector<vector<int>> &v) {
123     buildX(0, 0, N - 1, v);
124 }
125
126 int query(int aX, int bX, int aY, int bY) {
127     return queryX(0, 0, N - 1, aX, bX, aY, bY);
128 }
129
130 void update(int x, int y) {
131     updateX(0, 0, N - 1, x, y);
132 }
133 };

```

9.7 Minimum And Amount

```

1 // Description:
2 // Query - get minimum element in a range (l, r) inclusive
3 // and also the number of times it appears in that range
4 // Update - update element at position id to a value val
5
6 // Problem:
7 // https://codeforces.com/edu/course/2/lesson/4/1/practice/contest/273169/problem/C
8
9 // Complexity:
10 // O(log n) for both query and update
11
12 // How to use:
13 // Segtree seg = Segtree(n);
14 // seg.build(v);
15
16 #define pii pair<int, int>
17 #define mp make_pair
18 #define ff first
19 #define ss second
20
21 const int INF = 1e9+17;
22
23 typedef pii ftype;
24
25 struct Segtree {
26     vector<ftype> seg;
27     int n;
28     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]);
}

```

```

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

```

9.8 Lazy Addition To Segment

```

1 // Description:
2 // Query - get sum of elements from range (l, r)
   inclusive
3 // Update - add a value val to elementos from range (l, r) inclusive
4
5 // Problem:
6 // https://codeforces.com/edu/course/2/lesson/5/1/
   practice/contest/279634/problem/A
7
8 // Complexity:
9 // O(log n) for both query and update
10
11 // How to use:
12 // Segtree seg = Segtree(n);
13 // seg.build(v);
14
15 // Notes
16 // Change neutral element and f function to perform a
   different operation
17
18 const long long INF = 1e18+10;
19
20 typedef long long ftype;
21
22 struct Segtree {
23     vector<ftype> seg;
24     vector<ftype> lazy;
25     int n;
26     const ftype NEUTRAL = 0;
27     const ftype NEUTRAL_LAZY = -1; // change to -INF
   if there are negative numbers
28
29     Segtree(int n) {
30         int sz = 1;
31         while (sz < n) sz *= 2;
32         this->n = sz;
33
34         seg.assign(2*sz, NEUTRAL);
35         lazy.assign(2*sz, NEUTRAL_LAZY);
36     }
37
38     ftype apply_lazy(ftype a, ftype b, int len) {
39         if (b == NEUTRAL_LAZY) return a;
40         if (a == NEUTRAL_LAZY) return b * len;
41         else return a + b * len;
42     }
43
44     void propagate(int pos, int ini, int fim) {

```

```

45         if (ini == fim) {
46             return;
47         }
48
49         int e = 2*pos + 1;
50         int d = 2*pos + 2;
51         int m = ini + (fim - ini) / 2;
52
53         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
54         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
55
56         seg[e] = apply_lazy(seg[e], lazy[pos], m -
   ini + 1);
57         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
   m);
58
59         lazy[pos] = NEUTRAL_LAZY;
60     }
61
62     ftype f(ftype a, ftype b) {
63         return a + b;
64     }
65
66     ftype query(int pos, int ini, int fim, int p, int
   q) {
67         propagate(pos, ini, fim);
68
69         if (ini >= p && fim <= q) {
70             return seg[pos];
71         }
72
73         if (q < ini || p > fim) {
74             return NEUTRAL;
75         }
76
77         int e = 2*pos + 1;
78         int d = 2*pos + 2;
79         int m = ini + (fim - ini) / 2;
80
81         return f(query(e, ini, m, p, q), query(d, m +
   1, fim, p, q));
82     }
83
84     void update(int pos, int ini, int fim, int p, int
   q, int val) {
85         propagate(pos, ini, fim);
86
87         if (ini > q || fim < p) {
88             return;
89         }
90
91         if (ini >= p && fim <= q) {
92             lazy[pos] = apply_lazy(lazy[pos], val, 1)
   ;
93             seg[pos] = apply_lazy(seg[pos], val, fim
   - ini + 1);
94
95             return;
96         }
97
98         int e = 2*pos + 1;
99         int d = 2*pos + 2;
100         int m = ini + (fim - ini) / 2;
101
102         update(e, ini, m, p, q, val);
103         update(d, m + 1, fim, p, q, val);
104
105         seg[pos] = f(seg[e], seg[d]);
106     }
107
108     void build(int pos, int ini, int fim, vector<int>
   &v) {
109         if (ini == fim) {

```

```

110         if (ini < (int)v.size()) {
111             seg[pos] = v[ini];
112         }
113         return;
114     }
115
116     int e = 2*pos + 1;
117     int d = 2*pos + 2;
118     int m = ini + (fim - ini) / 2;
119
120     build(e, ini, m, v);
121     build(d, m + 1, fim, v);
122
123     seg[pos] = f(seg[e], seg[d]);
124 }
125
126 ftype query(int p, int q) {
127     return query(0, 0, n - 1, p, q);
128 }
129
130 void update(int p, int q, int val) {
131     update(0, 0, n - 1, p, q, val);
132 }
133
134 void build(vector<int> &v) {
135     build(0, 0, n - 1, v);
136 }
137
138 void debug() {
139     for (auto e : seg) {
140         cout << e << ' ';
141     }
142     cout << '\n';
143     for (auto e : lazy) {
144         cout << e << ' ';
145     }
146     cout << '\n';
147     cout << '\n';
148 }
149 };

```

9.9 Segment With Maximum Sum

```

1 // Description:
2 // Query - get sum of segment that is maximum among
3 // all segments
4 // E.g
5 // Array: 5 -4 4 3 -5
6 // Maximum segment sum: 8 because 5 + (-4) + 4 = 8
7 // Update - update element at position id to a value
8 // val
9 // Problem:
10 // https://codeforces.com/edu/course/2/lesson/4/2/
11 // practice/contest/273278/problem/A
12 // Complexity:
13 // O(log n) for both query and update
14 // How to use:
15 // Segtree seg = Segtree(n);
16 // seg.build(v);
17 // Notes
18 // The maximum segment sum can be a negative number
19 // In that case, taking zero elements is the best
20 // choice
21 // So we need to take the maximum between 0 and the
22 // query
23 // max(0LL, seg.query(0, n).max_seg)
24 using ll = long long;
25

```

```

26 typedef ll ftype_node;
27
28 struct Node {
29     ftype_node max_seg;
30     ftype_node pref;
31     ftype_node suf;
32     ftype_node sum;
33
34     Node(ftype_node max_seg, ftype_node pref,
35          ftype_node suf, ftype_node sum) : max_seg(max_seg),
36          pref(pref), suf(suf), sum(sum) {};
37 };
38
39 typedef Node ftype;
40
41 struct Segtree {
42     vector<ftype> seg;
43     int n;
44     const ftype NEUTRAL = Node(0, 0, 0, 0);
45
46     Segtree(int n) {
47         int sz = 1;
48         // potencia de dois mais proxima
49         while (sz < n) sz *= 2;
50         this->n = sz;
51
52         // numero de nos da seg
53         seg.assign(2*sz, NEUTRAL);
54     }
55
56     ftype f(ftype a, ftype b) {
57         ftype_node max_seg = max({a.max_seg, b.
58             max_seg, a.suf + b.pref});
59         ftype_node pref = max(a.pref, a.sum + b.pref);
60
61         ftype_node suf = max(b.suf, b.sum + a.suf);
62         ftype_node sum = a.sum + b.sum;
63
64         return Node(max_seg, pref, suf, sum);
65     }
66
67     ftype query(int pos, int ini, int fim, int p, int
68         q) {
69         if (ini >= p && fim <= q) {
70             return seg[pos];
71         }
72
73         if (q < ini || p > fim) {
74             return NEUTRAL;
75         }
76
77         int e = 2*pos + 1;
78         int d = 2*pos + 2;
79         int m = ini + (fim - ini) / 2;
80
81         return f(query(e, ini, m, p, q), query(d, m +
82             1, fim, p, q));
83     }
84
85     void update(int pos, int ini, int fim, int id,
86         int val) {
87         if (ini > id || fim < id) {
88             return;
89         }
90
91         if (ini == id && fim == id) {
92             seg[pos] = Node(val, val, val, val);
93
94             return;
95         }
96
97         int e = 2*pos + 1;
98         int d = 2*pos + 2;
99

```

```

92         int m = ini + (fim - ini) / 2;
93
94         update(e, ini, m, id, val);
95         update(d, m + 1, fim, id, val);
96
97         seg[pos] = f(seg[e], seg[d]);
98     }
99
100 void build(int pos, int ini, int fim, vector<int>
    &v) {
101     if (ini == fim) {
102         // se a posição existir no array original
103         // seg tamanho potencia de dois
104         if (ini < (int)v.size()) {
105             seg[pos] = Node(v[ini], v[ini], v[ini]
106 ], v[ini]);
107         }
108         return;
109     }
110
111     int e = 2*pos + 1;
112     int d = 2*pos + 2;
113     int m = ini + (fim - ini) / 2;
114
115     build(e, ini, m, v);
116     build(d, m + 1, fim, v);
117
118     seg[pos] = f(seg[e], seg[d]);
119 }
120
121 ftype query(int p, int q) {
122     return query(0, 0, n - 1, p, q);
123 }
124
125 void update(int id, int val) {
126     update(0, 0, n - 1, id, val);
127 }
128
129 void build(vector<int> &v) {
130     build(0, 0, n - 1, v);
131 }
132
133 void debug() {
134     for (auto e : seg) {
135         cout << e.max_seg << ' ' << e.pref << ' '
136 << e.suf << ' ' << e.sum << '\n';
137     }
138 };

```

```

17 // Change neutral element and f function to perform a
    different operation
18
19 // If you want to change the operations to point
    query and range update
20 // Use the same segtree, but perform the following
    operations
21 // Query - seg.query(0, id);
22 // Update - seg.update(l, v); seg.update(r + 1, -v);
23
24 typedef long long ftype;
25
26 struct Segtree {
27     vector<ftype> seg;
28     int n;
29     const ftype NEUTRAL = 0;
30
31     Segtree(int n) {
32         int sz = 1;
33         while (sz < n) sz *= 2;
34         this->n = sz;
35
36         seg.assign(2*sz, NEUTRAL);
37     }
38
39     ftype f(ftype a, ftype b) {
40         return a + b;
41     }
42
43     ftype query(int pos, int ini, int fim, int p, int
        q) {
44         if (ini >= p && fim <= q) {
45             return seg[pos];
46         }
47
48         if (q < ini || p > fim) {
49             return NEUTRAL;
50         }
51
52         int e = 2*pos + 1;
53         int d = 2*pos + 2;
54         int m = ini + (fim - ini) / 2;
55
56         return f(query(e, ini, m, p, q), query(d, m +
57 1, fim, p, q));
58     }
59
60     void update(int pos, int ini, int fim, int id,
        int val) {
61         if (ini > id || fim < id) {
62             return;
63         }
64
65         if (ini == id && fim == id) {
66             seg[pos] = val;
67         }
68
69         return;
70
71         int e = 2*pos + 1;
72         int d = 2*pos + 2;
73         int m = ini + (fim - ini) / 2;
74
75         update(e, ini, m, id, val);
76         update(d, m + 1, fim, id, val);
77
78         seg[pos] = f(seg[e], seg[d]);
79     }
80
81     void build(int pos, int ini, int fim, vector<int>
        &v) {
82         if (ini == fim) {
83             if (ini < (int)v.size()) {

```

9.10 Range Query Point Update

```

1 // Description:
2 // Indexed at zero
3 // Query - get sum of elements from range (l, r)
    inclusive
4 // Update - update element at position id to a value
    val
5
6 // Problem:
7 // https://codeforces.com/edu/course/2/lesson/4/1/
    practice/contest/273169/problem/B
8
9 // Complexity:
10 // O(log n) for both query and update
11
12 // How to use:
13 // Segtree seg = Segtree(n);
14 // seg.build(v);
15
16 // Notes

```

```

83         seg[pos] = v[ini];
84     }
85     return;
86 }
87
88 int e = 2*pos + 1;
89 int d = 2*pos + 2;
90 int m = ini + (fim - ini) / 2;
91
92 build(e, ini, m, v);
93 build(d, m + 1, fim, v);
94
95 seg[pos] = f(seg[e], seg[d]);
96 }
97
98 ftype query(int p, int q) {
99     return query(0, 0, n - 1, p, q);
100 }
101
102 void update(int id, int val) {
103     update(0, 0, n - 1, id, val);
104 }
105
106 void build(vector<int> &v) {
107     build(0, 0, n - 1, v);
108 }
109
110 void debug() {
111     for (auto e : seg) {
112         cout << e << ' ';
113     }
114     cout << '\n';
115 }
116 };

```

9.11 Lazy Assignment To Segment

```

1  const long long INF = 1e18+10;
2
3  typedef long long ftype;
4
5  struct Segtree {
6      vector<ftype> seg;
7      vector<ftype> lazy;
8      int n;
9      const ftype NEUTRAL = 0;
10     const ftype NEUTRAL_LAZY = -1; // Change to -INF
    if there are negative numbers
11
12     Segtree(int n) {
13         int sz = 1;
14         // potencia de dois mais proxima
15         while (sz < n) sz *= 2;
16         this->n = sz;
17
18         // numero de nos da seg
19         seg.assign(2*sz, NEUTRAL);
20         lazy.assign(2*sz, NEUTRAL_LAZY);
21     }
22
23     ftype apply_lazy(ftype a, ftype b, int len) {
24         if (b == NEUTRAL_LAZY) return a;
25         if (a == NEUTRAL_LAZY) return b * len;
26         else return b * len;
27     }
28
29     void propagate(int pos, int ini, int fim) {
30         if (ini == fim) {
31             return;
32         }
33
34         int e = 2*pos + 1;
35         int d = 2*pos + 2;

```

```

36         int m = ini + (fim - ini) / 2;
37
38         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
39         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
40
41         seg[e] = apply_lazy(seg[e], lazy[pos], m -
    ini + 1);
42         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
    m);
43
44         lazy[pos] = NEUTRAL_LAZY;
45     }
46
47     ftype f(ftype a, ftype b) {
48         return a + b;
49     }
50
51     ftype query(int pos, int ini, int fim, int p, int
    q) {
52         propagate(pos, ini, fim);
53
54         if (ini >= p && fim <= q) {
55             return seg[pos];
56         }
57
58         if (q < ini || p > fim) {
59             return NEUTRAL;
60         }
61
62         int e = 2*pos + 1;
63         int d = 2*pos + 2;
64         int m = ini + (fim - ini) / 2;
65
66         return f(query(e, ini, m, p, q), query(d, m +
    1, fim, p, q));
67     }
68
69     void update(int pos, int ini, int fim, int p, int
    q, int val) {
70         propagate(pos, ini, fim);
71
72         if (ini > q || fim < p) {
73             return;
74         }
75
76         if (ini >= p && fim <= q) {
77             lazy[pos] = apply_lazy(lazy[pos], val, 1)
    ;
78             seg[pos] = apply_lazy(seg[pos], val, fim
    - ini + 1);
79
80             return;
81         }
82
83         int e = 2*pos + 1;
84         int d = 2*pos + 2;
85         int m = ini + (fim - ini) / 2;
86
87         update(e, ini, m, p, q, val);
88         update(d, m + 1, fim, p, q, val);
89
90         seg[pos] = f(seg[e], seg[d]);
91     }
92
93     void build(int pos, int ini, int fim, vector<int>
    &v) {
94         if (ini == fim) {
95             // se a posição existir no array original
96             // seg tamanho potencia de dois
97             if (ini < (int)v.size()) {
98                 seg[pos] = v[ini];
99             }
100             return;

```



```

101     }
102
103     int e = 2*pos + 1;
104     int d = 2*pos + 2;
105     int m = ini + (fim - ini) / 2;
106
107     build(e, ini, m, v);
108     build(d, m + 1, fim, v);
109
110     seg[pos] = f(seg[e], seg[d]);
111 }
112
113 ftype query(int p, int q) {
114     return query(0, 0, n - 1, p, q);
115 }
116
117 void update(int p, int q, int val) {
118     update(0, 0, n - 1, p, q, val);
119 }
120
121 void build(vector<int> &v) {
122     build(0, 0, n - 1, v);
123 }
124
125 void debug() {
126     for (auto e : seg) {
127         cout << e << ' ';
128     }
129     cout << '\n';
130     for (auto e : lazy) {
131         cout << e << ' ';
132     }
133     cout << '\n';
134     cout << '\n';
135 }
136 };

```

9.12 Lazy Dynamic Implicit Sparse

```

1 // Description:
2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
5 // stored in an array
6 // and the queries need to be answered online so we
7 // can't sort the nodes and compress them
8 // we create nodes only when they are needed so there
9 // 'll be (Q*log(MAX)) nodes
10 // where Q is the number of queries and MAX is the
11 // maximum index a node can assume
12
13 // Query - get sum of elements from range (l, r)
14 // inclusive
15 // Update - update element at position id to a value
16 // val
17
18 // Problem:
19 // https://oj.uz/problem/view/IZh012_apple
20
21 // Complexity:
22 // O(log n) for both query and update
23
24 // How to use:
25 // MAX is the maximum index a node can assume
26 // Create a default null node
27 // Create a node to be the root of the segtree
28
29 // Segtree seg = Segtree(MAX);
30
31 const int MAX = 1e9+10;
32 const long long INF = 1e18+10;
33
34 typedef long long ftype;

```

```

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

```

```

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

```

9.13 Persistent

```

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

```

```

26 typedef long long ftype;
27
28 struct Segtree {
29     vector<ftype> seg, d, e;
30     const ftype NEUTRAL = 0;
31     int n;
32
33     Segtree(int n) {
34         this->n = n;
35     }
36
37     ftype f(ftype a, ftype b) {
38         return a + b;
39     }
40
41     ftype create() {
42         seg.push_back(0);
43         e.push_back(0);
44         d.push_back(0);
45         return seg.size() - 1;
46     }
47
48     ftype query(int pos, int ini, int fim, int p, int
49 q) {
50         if (q < ini || p > fim) return NEUTRAL;
51         if (pos == 0) return 0;
52         if (p <= ini && fim <= q) return seg[pos];
53         int m = (ini + fim) >> 1;
54         return f(query(e[pos], ini, m, p, q), query(d
55 [pos], m + 1, fim, p, q));
56
57     }
58
59     int update(int pos, int ini, int fim, int id, int
60 val) {
61         int novo = create();
62
63         seg[novo] = seg[pos];
64         e[novo] = e[pos];
65         d[novo] = d[pos];
66
67         if (ini == fim) {
68             seg[novo] = val;
69             return novo;
70         }
71
72         int m = (ini + fim) >> 1;
73
74         if (id <= m) e[novo] = update(e[novo], ini, m
75 , id, val);
76         else d[novo] = update(d[novo], m + 1, fim, id
77 , val);
78
79         seg[novo] = f(seg[e[novo]], seg[d[novo]]);
80
81         return novo;
82     }
83
84     ftype query(int pos, int p, int q) {
85         return query(pos, 1, n, p, q);
86     }
87
88     int update(int pos, int id, int val) {
89         return update(pos, 1, n, id, val);
90     }
91 };

```