



Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1 Math	2	4 Strings	7
1.1 Ceil	2	4.1 Kmp	7
1.2 To Decimal	2	4.2 Generate All Permutations	7
1.3 Matrix Exponentiation	2	4.3 Generate All Sequences Length K	8
1.4 Crt	3	4.4 Lcs	8
1.5 Binary To Decimal	3	4.5 Trie	8
1.6 Fast Exponentiation	3	4.6 Z-function	8
1.7 Linear Diophantine Equation	3	5 Misc	9
1.8 Function Root	4	5.1 Split	9
1.9 Sieve Of Eratosthenes	4	5.2 Int128	9
1.10 Horner Algorithm	4	6 Graphs	9
1.11 Multiplicative Inverse	4	6.1 Centroid Find	9
1.12 Representation Arbitrary Base	4	6.2 Bipartite	9
1.13 Divisors	5	6.3 Prim	10
1.14 Check If Bit Is On	5	6.4 Ford Fulkerson Edmonds Karp	10
1.15 Prime Factors	5	6.5 Floyd Warshall	10
2 DP	5	6.6 Lca	11
2.1 Knapsack With Index	5	6.7 Bellman Ford	11
2.2 Substr Palindrome	5	6.8 Dinic	12
2.3 Edit Distance	5	6.9 2sat	13
2.4 Knapsack	6	6.10 Find Cycle	15
2.5 Digits	6	6.11 Cycle Path Recovery	15
2.6 Coins	6	6.12 Centroid Decomposition	15
2.7 Minimum Coin Change	6	6.13 Tarjan Bridge	16
2.8 Kadane	6	6.14 Small To Large	16
3 Template	7	6.15 Tree Diameter	17
3.1 Template	7	6.16 Dijkstra	17
3.2 Template Clean	7	6.17 Kruskall	18
		7 Geometry	18
		7.1 2d	18

8	Algorithms	20
8.1	Lis	20
8.2	Delta-encoding	21
8.3	Subsets	21
8.4	Binary Search Last True	21
8.5	Ternary Search	21
8.6	Binary Search First True	21
8.7	Biggest K	21
9	Data Structures	22
9.1	Ordered Set	22
9.2	Priority Queue	22
9.3	Dsu	22
9.4	Two Sets	23
9.5	Dynamic Implicit Sparse	24
9.6	Segtree2d	24
9.7	Minimum And Amount	25
9.8	Lazy Addition To Segment	26
9.9	Segment With Maximum Sum	27
9.10	Range Query Point Update	28
9.11	Lazy Assignment To Segment	29
9.12	Lazy Dynamic Implicit Sparse	30
9.13	Persistent	31

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         n *= base;
10        n += digits.find(c);
11    }
12
13    return n;
14 }
```

1.3 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
24
25 // Adding another column, there are:
26 // 3 ways to go from 2 equal to 2 equal
27 // 2 ways to go from 2 equal to 3 distinct
28 // 2 ways to go from 3 distinct to 2 equal
29 // 2 ways to go from 3 distinct to 3 distinct
30
31 // So we star with matrix 6 6 and multiply it by the
32 // transition 3 2 and get 18 12
33 //
34 //      2 2      12 12
35 // the we can exponentiate this matrix to find the
36 // nth column
37
38 // Problem:
39 // https://cses.fi/problemset/task/1722/
40
41 // Complexity:
42 // O(log n)
```

```
38 // How to use:
39 // vector<vector<ll>> v = {{1, 1}, {1, 0}};
40 // Matriz transition = Matriz(v);
41 // cout << fexp(transition, n)[0][1] << '\n';
42
43 using ll = long long;
44
45 const int MOD = 1e9+7;
46
47 struct Matriz{
48     vector<vector<ll>> mat;
49     int rows, columns;
50
51     vector<ll> operator[](int i){
52         return mat[i];
53     }
54
55     Matriz(vector<vector<ll>>& matriz){
56         mat = matriz;
57         rows = mat.size();
58         columns = mat[0].size();
59     }
60
61     Matriz(int row, int column, bool identity=false){
62         rows = row; columns = column;
63         mat.assign(rows, vector<ll>(columns, 0));
64         if(identity) {
65             for(int i = 0; i < min(rows, columns); i
66 ++){
67                 mat[i][i] = 1;
68             }
69         }
70
71     Matriz operator * (Matriz a) {
72         assert(columns == a.rows);
73         vector<vector<ll>> resp(rows, vector<ll>(a.
74 columns, 0));
75
76         for(int i = 0; i < rows; i++){
77             for(int j = 0; j < a.columns; j++){
78                 for(int k = 0; k < a.rows; k++){
79                     resp[i][j] = (resp[i][j] + (mat[i
80 ][k] * 1LL * a[k][j]) % MOD) % MOD;
81                 }
82             }
83         }
84         return Matriz(resp);
85     }
86
87     Matriz operator + (Matriz a) {
88         assert(rows == a.rows && columns == a.columns
89 );
90         vector<vector<ll>> resp(rows, vector<ll>(
91 columns, 0));
92         for(int i = 0; i < rows; i++){
93             for(int j = 0; j < columns; j++){
94                 resp[i][j] = (resp[i][j] + mat[i][j]
95 + a[i][j]) % MOD;
96             }
97         }
98         return Matriz(resp);
99     }
100 };
101
102 Matriz fexp(Matriz base, ll exponent){
103     Matriz result = Matriz(base.rows, base.columns, 1);
104     while(exponent > 0){
105         if(exponent & 1LL) result = result * base;
106         base = base * base;
107         exponent = exponent >> 1;
108     }
109     return result;
110 }
```

```
105 }
```

1.4 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 }
```

1.5 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 }
```

1.6 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 }
```

1.7 Linear Diophantine Equation

```
1 // int a, b, c, x1, x2, y1, y2; cin >> a >> b >> c >>
2 // x1 >> x2 >> y1 >> y2;
3 // int ans = -1;
4 // if (a == 0 && b == 0) {
5 //     if (c != 0) ans = 0;
6 //     else ans = (x2 - x1 + 1) * (y2 - y1 + 1);
```

```
6 // }
7 // else if (a == 0) {
8 //     if (c % b == 0 && y1 <= c / b && y2 >= c / b)
9 //         ans = (x2 - x1 + 1);
10 //     else ans = 0;
11 // }
12 // else if (b == 0) {
13 //     if (c % a == 0 && x1 <= c / a && x2 >= c / a)
14 //         ans = (y2 - y1 + 1);
15 //     else ans = 0;
16 // }
17 // Careful when a or b are negative or zero
18 // if (ans == -1) ans = find_all_solutions(a, b, c,
19 //     x1, x2, y1, y2);
20 // cout << ans << '\n';
21 // Problems:
22 // https://www.spoj.com/problems/CEQU/
23 // http://codeforces.com/problemsets/acmsguru/problem
24 // 99999/106
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
40 // passed as reference
41 // minx <= x <= maxx miny <= y <= maxy
42 bool find_any_solution(int a, int b, int c, int &x0,
43     int &y0, int &g) {
44     g = gcd(abs(a), abs(b), x0, y0);
45     if (c % g) {
46         return false;
47     }
48     x0 *= c / g;
49     y0 *= c / g;
50     if (a < 0) x0 = -x0;
51     if (b < 0) y0 = -y0;
52     return true;
53 }
54 void shift_solution(int &x, int &y, int a, int b,
55     int cnt) {
56     x += cnt * b;
57     y -= cnt * a;
58 }
59 // return number of solutions in the interval
60 int find_all_solutions(int a, int b, int c, int minx,
61     int maxx, int miny, int maxy) {
62     int x, y, g;
63     if (!find_any_solution(a, b, c, x, y, g))
64         return 0;
65     a /= g;
66     b /= g;
67
68     int sign_a = a > 0 ? +1 : -1;
69     int sign_b = b > 0 ? +1 : -1;
70
71     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 }

```

1.8 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 *
7          tan(x) + t * x * x + u;
8  }
9
10 ld root(ld a, ld b) {
11     while (b - a >= EPS1) {
12         ld c = (a + b) / 2.0;
13         ld y = f(c);
14
15         if (y < 0) b = c;
16         else a = c;
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 <<
24         setprecision(4) << ans << '\n';
25     else cout << "No solution\n";
26
27     return 0;
28 }

```

1.9 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 }

```

1.10 Horner Algorithm

```

1  // Description:
2  // Evaluates y = f(x)
3
4  // Problem:
5  // https://onlinejudge.org/index.php?option=
6  // com_onlinejudge&Itemid=8&page=show_problem&
7  // problem=439
8
9  // Complexity:
10 // O(n)
11
12 using polynomial = std::vector<int>;
13
14 polynomial p {6, -5, 2}; // p(x) = x^2 - 5x + 6;
15
16 int degree(const polynomial& p) {
17     return p.size() - 1;
18 }
19
20 int evaluate(const polynomial& p, int x) {
21     int y = 0, N = degree(p);
22
23     for (int i = N; i >= 0; --i) {
24         y *= x;
25         y += p[i];
26     }
27
28     return y;
29 }

```

1.11 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 }

```

1.12 Representation Arbitrary Base

```

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

```

1.13 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 }

```

1.14 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 }

```

1.15 Prime Factors

```

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

```

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    int res = K[n][W];
18    cout<< res << endl;
19
20    w = W;
21    for (i = n; i > 0 && res > 0; i--) {
22        if (res == K[i - 1][w])
23            continue;
24        else {
25            cout<<" "<<wt[i - 1] ;
26            res = res - val[i - 1];
27            w = w - wt[i - 1];
28        }
29    }
30 }
31
32 int main()
33 {
34     int val[] = { 60, 100, 120 };
35     int wt[] = { 10, 20, 30 };
36     int W = 50;
37     int n = sizeof(val) / sizeof(val[0]);
38
39     knapsack(W, wt, val, n);
40
41     return 0;
42 }

```

2.2 Substr Palindrome

```

1 // êvoc deve informar se a substring de S formada
  pelos elementos entre os indices i e j
2 // é um palindromo ou ãno.
3
4 char s[MAX];
5 int calculado[MAX][MAX]; // inciado 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 or 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
  soma seja a maior possivel
2 // nesse caso vc precisa multiplicar exatamente 1
  elemento da subsequencia
3 // e achar a maior soma com isso
4
5 int n, x, arr[MAX], tab[MAX][2]; // tab[maior
  resposta no intervalo][foi multiplicado ou ão]
6
7 int dp(int i, bool mult) {
8     if (i == n-1) {
9         if (!mult) return arr[n-1]*x;
10        return arr[n-1];
11    }
12    if (tab[i][mult] != -1) return tab[i][mult];
13
14    int res;
15
16    if (mult) {
17        res = max(arr[i], arr[i] + dp(i+1, 1));
18    }
19    else {
20        res = max({
21            arr[i]*x,
22            arr[i]*x + dp(i+1, 1),
23            arr[i] + dp(i+1, 0)
24        });
25    }
26
27    return tab[i][mult] = res;
28 }
29
30 int main() {
31
32    memset(tab, -1, sizeof(tab));
33
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);
  cin.tie(NULL);

```

```

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

```

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 +
11             set[i], n, k - 1);
12         ans.insert(ans.end(), aux.begin(), aux.end());
13     };
14     // for (auto e : aux) ans.push_back(e);
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     }
39 }

```

```

35     }
36
37     else if (LCS_table[i - 1][j] > LCS_table[i][j -
38         1])
39         i--;
40     else
41         j--;
42
43     return lcsAlgo;
44 }

```

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    if(palTemp.size() > 0)
18        ans.push_back(palTemp);
19
20    return ans;
21 }
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 void print(__int128 x) {
15     if (x < 0) {
16         putchar('-');
17         x = -x;
18     }
19     if (x > 9) print(x / 10);
20     putchar(x % 10 + '0');
21 }

```

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 // Complexity:
9 // O(n)
10 // How to use:
11 // get_subtree_size(0);
12 // cout << get_centroid(0) + 1 << endl;
13
14 int n;
15 vector<int> adj[MAX];
16 int subtree_size[MAX];
17
18 int get_subtree_size(int node, int par = -1) {
19     int &res = subtree_size[node];
20     res = 1;
21     for (int i : adj[node]) {
22         if (i == par) continue;
23         res += get_subtree_size(i, node);
24     }
25     return res;
26 }
27
28 int get_centroid(int node, int par = -1) {
29     for (int i : adj[node]) {
30         if (i == par) continue;
31
32         if (subtree_size[i] * 2 > n) { return
33             get_centroid(i, node); }
34     }
35     return node;
36 }
37
38 int main() {
39     cin >> n;
40     for (int i = 0; i < n - 1; i++) {
41         int u, v; cin >> u >> v;
42         u--; v--;
43         adj[u].push_back(v);
44         adj[v].push_back(u);
45     }
46
47     get_subtree_size(0);
48     cout << get_centroid(0) + 1 << endl;
49 }
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        }
21    }
22 }

```

```

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
23         if (min_e[v].w == INF) {
24             cout << "No MST!" << endl;
25             exit(0);
26         }
27
28         selected[v] = true;
29         total_weight += min_e[v].w;
30         if (min_e[v].to != -1)
31             cout << v << " " << min_e[v].to << endl;
32
33         for (int to = 0; to < n; ++to) {
34             if (adj[v][to] < min_e[to].w)
35                 min_e[to] = {adj[v][to], v};
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     return flow;
52 }
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 = 0x3f3f3f3f3f3f3f3fLL;
8
9 ll dist[MAX][MAX];
10 int n;
11
12 void floyd_warshall() {
13     for (int i = 0; i < n; i++) {
14         for (int j = 0; j < n; j++) {
15             if (i == j) dist[i][j] = 0;
16             else if (!dist[i][j]) dist[i][j] = INF;
17         }
18     }

```

```

19     for (int k = 0; k < n; k++) {
20         for (int i = 0; i < n; i++) {
21             for (int j = 0; j < n; j++) {
22                 // trata o caso no qual o grafo tem
23                 arestas com peso negativo
24                 if (dist[i][k] < INF && dist[k][j] <
25                     INF){
26                     dist[i][j] = min(dist[i][j], dist
27                     [i][k] + dist[k][j]);
28                 }
29             }
30     }

```

6.6 Lca

```

1 // Description:
2 // Find the lowest common ancestor between two nodes
  in a tree
3
4 // Problem:
5 // https://cses.fi/problemset/task/1135
6
7 // Complexity:
8 // O(log n)
9
10 // How to use:
11 // preprocess();
12 // lca(a, b);
13
14 // Notes
15 // To calculate the distance between two nodes use
  the following formula
16 // level_peso[a] + level_peso[b] - 2*level_peso[lca(a
  , b)]
17
18 const int MAX = 2e5+10;
19 const int BITS = 30;
20
21 vector<pii> adj[MAX];
22 vector<bool> visited(MAX);
23
24 int up[MAX][BITS + 1];
25 int level[MAX];
26 int level_peso[MAX];
27
28 void find_level() {
29     queue<pii> q;
30
31     q.push(mp(1, 0));
32     visited[1] = true;
33
34     while (!q.empty()) {
35         auto [v, depth] = q.front();
36         q.pop();
37         level[v] = depth;
38
39         for (auto [u,d] : adj[v]) {
40             if (!visited[u]) {
41                 visited[u] = true;
42                 up[u][0] = v;
43                 q.push(mp(u, depth + 1));
44             }
45         }
46     }
47 }
48
49 void find_level_peso() {
50     queue<pii> q;
51
52     q.push(mp(1, 0));

```

```

53     visited[1] = true;
54
55     while (!q.empty()) {
56         auto [v, depth] = q.front();
57         q.pop();
58         level_peso[v] = depth;
59
60         for (auto [u,d] : adj[v]) {
61             if (!visited[u]) {
62                 visited[u] = true;
63                 up[u][0] = v;
64                 q.push(mp(u, depth + d));
65             }
66         }
67     }
68 }
69
70 int lca(int a, int b) {
71     // get the nodes to the same level
72     int mn = min(level[a], level[b]);
73
74     for (int j = 0; j <= BITS; j++) {
75         if (a != -1 && ((level[a] - mn) & (1 << j))) a
76         = up[a][j];
77         if (b != -1 && ((level[b] - mn) & (1 << j))) b
78         = up[b][j];
79     }
80
81     // special case
82     if (a == b) return a;
83
84     // binary search
85     for (int j = BITS; j >= 0; j--) {
86         if (up[a][j] != up[b][j]) {
87             a = up[a][j];
88             b = up[b][j];
89         }
90     }
91     return up[a][0];
92 }
93
94 void preprocess() {
95     visited = vector<bool>(MAX, false);
96     find_level();
97     visited = vector<bool>(MAX, false);
98     find_level_peso();
99
100     for (int j = 1; j <= BITS; j++) {
101         for (int i = 1; i <= n; i++) {
102             if (up[i][j - 1] != -1) up[i][j] = up[up[i][j -
103             1]][j - 1];
104         }
105     }

```

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
3 // network. A network is a graph with a single
4 // source vertex and a single sink vertex in which
5 // each edge has a capacity
6
7 // Problem:
8 // https://codeforces.com/gym/103708/problem/J
9
10 // Complexity:
11 //  $O(V^2 * E)$  where V is the number of vertex and E
12 // is the number of edges
13
14 // Unit network
15 // A unit network is a network in which for any
16 // vertex except source and sink either incoming or
17 // outgoing edge is unique and has unit capacity (
18 // matching problem).
19
20 // Complexity on unit networks:  $O(E * \sqrt{V})$ 
21
22 // Unity capacity networks
23 // A more generic settings when all edges have unit
24 // capacities, but the number of incoming and
25 // outgoing edges is unbounded
26
27 // Complexity on unity capacity networks:  $O(E * \sqrt{E})$ 
28
29 // How to use:
30 // Dinic dinic = Dinic(num_vertex, source, sink);
31 // dinic.add_edge(vertex1, vertex2, capacity);
32 // cout << dinic.max_flow() << '\n';
33
34 #include <bits/stdc++.h>
35
36 #define pb push_back
37 #define mp make_pair
38 #define pii pair<int, int>
39 #define ff first
40 #define ss second
41 #define ll long long
42
43 using namespace std;
44
45 const ll INF = 1e18+10;
46
47 struct Edge {
48     int from;
49     int to;
50     ll capacity;
51     ll flow;
52     Edge* residual;
53
54     Edge() {}
55
56     Edge(int from, int to, ll capacity) : from(from),
57     to(to), capacity(capacity) {
58         flow = 0;
59     }
60
61     ll get_capacity() {
62         return capacity - flow;
63     }
64
65     ll get_flow() {
66         return flow;
67     }
68
69 
```

```

57 void augment(ll bottleneck) {
58     flow += bottleneck;
59     residual->flow -= bottleneck;
60 }
61
62 void reverse(ll bottleneck) {
63     flow -= bottleneck;
64     residual->flow += bottleneck;
65 }
66
67 bool operator<(const Edge& e) const {
68     return true;
69 }
70 };
71
72 struct Dinic {
73     int source;
74     int sink;
75     int nodes;
76     ll flow;
77     vector<vector<Edge*>> adj;
78     vector<int> level;
79     vector<int> next;
80     vector<int> reach;
81     vector<bool> visited;
82     vector<vector<int>> path;
83
84     Dinic(int source, int sink, int nodes) : source(
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);
91         Edge* e2 = new Edge(to, from, 0);
92         // Edge* e2 = new Edge(to, from, capacity);
93         e1->residual = e2;
94         e2->residual = e1;
95         adj[from].pb(e1);
96         adj[to].pb(e2);
97     }
98
99     bool bfs() {
100         level.assign(nodes + 1, -1);
101         queue<int> q;
102         q.push(source);
103         level[source] = 0;
104
105         while (!q.empty()) {
106             int node = q.front();
107             q.pop();
108
109             for (auto e : adj[node]) {
110                 if (level[e->to] == -1 && e->
111                 get_capacity() > 0) {
112                     level[e->to] = level[e->from] +
113                     1;
114                     q.push(e->to);
115                 }
116             }
117         }
118
119         return level[sink] != -1;
120     }
121
122     ll dfs(int v, ll flow) {
123         if (v == sink)
124             return flow;
125
126         int sz = adj[v].size();
127         for (int i = next[v]; i < sz; i++) {
128             Edge* e = adj[v][i];
129             if (level[e->to] == level[e->from] + 1 &&

```

```

127     e->get_capacity() > 0) {
128         ll bottleneck = dfs(e->to, min(flow,
129     e->get_capacity()));
130         if (bottleneck > 0) {
131             e->augment(bottleneck);
132             return bottleneck;
133         }
134         next[v] = i + 1;
135     }
136     return 0;
137 }
138
139 ll max_flow() {
140     flow = 0;
141     while(bfs()) {
142         next.assign(nodes + 1, 0);
143         ll sent = -1;
144         while (sent != 0) {
145             sent = dfs(source, INF);
146             flow += sent;
147         }
148     }
149     return flow;
150 }
151
152 void reachable(int v) {
153     visited[v] = true;
154
155     for (auto e : adj[v]) {
156         if (!visited[e->to] && e->get_capacity()
157 > 0) {
158             reach.pb(e->to);
159             visited[e->to] = true;
160             reachable(e->to);
161         }
162     }
163 }
164
165 void print_min_cut() {
166     reach.clear();
167     visited.assign(nodes + 1, false);
168     reach.pb(source);
169     reachable(source);
170
171     for (auto v : reach) {
172         for (auto e : adj[v]) {
173             if (!visited[e->to] && e->
174 get_capacity() == 0) {
175                 cout << e->from << ' ' << e->to
176 << '\n';
177             }
178         }
179     }
180
181 ll build_path(int v, int id, ll flow) {
182     visited[v] = true;
183     if (v == sink) {
184         return flow;
185     }
186     for (auto e : adj[v]) {
187         if (!visited[e->to] && e->get_flow() > 0)
188 {
189             visited[e->to] = true;
190             ll bottleneck = build_path(e->to, id,
191 min(flow, e->get_flow()));
192             if (bottleneck > 0) {
193                 path[id].pb(e->to);
194                 e->reverse(bottleneck);
195             }
196         }
197     }
198     return bottleneck;
199 }
200
201 void print_flow_path() {
202     path.clear();
203     ll sent = -1;
204     int id = -1;
205     while (sent != 0) {
206         visited.assign(nodes + 1, false);
207         path.pb(vector<int>{});
208         sent = build_path(source, ++id, INF);
209         path[id].pb(source);
210     }
211     path.pop_back();
212
213     for (int i = 0; i < id; i++) {
214         cout << path[i].size() << '\n';
215         reverse(path[i].begin(), path[i].end());
216         for (auto e : path[i]) {
217             cout << e << ' ';
218         }
219         cout << '\n';
220     }
221 }
222 };
223
224 int main() {
225     ios::sync_with_stdio(false);
226     cin.tie(NULL);
227
228     int n, m; cin >> n >> m;
229
230     Dinic dinic = Dinic(1, n, n);
231
232     for (int i = 1; i <= m; i++) {
233         int v, u; cin >> v >> u;
234         dinic.add_edge(v, u, 1);
235     }
236
237     cout << dinic.max_flow() << '\n';
238     // dinic.print_min_cut();
239     // dinic.print_flow_path();
240
241     return 0;
242 }

```

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         }
126
127         for (int i = 1; i <= nodes; i++) {
128             if (scc[i] == scc[i + nodes]) return
129                 false;
130         }
131
132         return true;
133     }
134
135     int find_value(int e, vector<int> &ans) {
136         if (e > nodes && ans[e - nodes] != 2) return
137             !ans[e - nodes];
138         if (e <= nodes && ans[e + nodes] != 2) return
139             !ans[e + nodes];
140         return 0;
141     }
142
143     vector<int> find_ans() {
144         condensed.resize(component + 1);
145
146         for (int i = 1; i <= 2 * nodes; i++) {
147             for (auto u : adj[i]) {
148                 if (scc[i] != scc[u]) condensed[scc[i]
149                     ]].pb(scc[u]);
150             }
151         }
152
153         visited.assign(component + 1, false);
154
155         for (int i = 1; i <= component; i++) {
156             if (!visited[i]) topological_order(i);
157         }
158
159         reverse(order.begin(), order.end());
160
161         // 0 - false
162         // 1 - true
163         // 2 - no value yet
164         vector<int> ans(2 * nodes + 1, 2);

```

```

158     vector<vector<int>> belong(component + 1);
159
160     for (int i = 1; i <= 2 * nodes; i++) {
161         belong[scc[i]].pb(i);
162     }
163
164     for (auto p : order) {
165         for (auto e : belong[p]) {
166             ans[e] = find_value(e, ans);
167         }
168     }
169
170     return ans;
171 }
172 };
173
174 int main() {
175     ios::sync_with_stdio(false);
176     cin.tie(NULL);
177
178     int n, m; cin >> n >> m;
179
180     SAT sat = SAT(m);
181
182     for (int i = 0; i < n; i++) {
183         char op1, op2; int a, b; cin >> op1 >> a >>
184         op2 >> b;
185         if (op1 == '+' && op2 == '+') sat.add_or(a, b);
186         if (op1 == '-' && op2 == '-') sat.add_or(sat.get_not(a), sat.get_not(b));
187         if (op1 == '+' && op2 == '-') sat.add_or(a, sat.get_not(b));
188         if (op1 == '-' && op2 == '+') sat.add_or(sat.get_not(a), b);
189     }
190
191     if (!sat.is_possible()) cout << "IMPOSSIBLE\n";
192     else {
193         vector<int> ans = sat.find_ans();
194         for (int i = 1; i <= m; i++) {
195             cout << (ans[i] == 1 ? '+' : '-') << ' ';
196         }
197         cout << '\n';
198     }
199
200     return 0;
201 }

```

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     visited.reset();
26
27     for (int u = 1; u <= N; ++u){
28         path.clear();
29         if (not visited[u] and dfs(u, -1))
30             return true;
31     }
32
33     return false;
34 }
35
36 }

```

6.11 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 = parent[v])
40             cycle.push_back(v);
41         cycle.push_back(cycle_start);
42         reverse(cycle.begin(), cycle.end());
43
44         cout << "Cycle found: ";
45         for (int v : cycle)
46             cout << v << " ";
47         cout << endl;
48     }
49 }

```

6.12 Centroid Decomposition

```

1  int n;

```



```

2  vector<set<int>> adj;
3  vector<char> ans;
4
5  vector<bool> removed;
6
7  vector<int> subtree_size;
8
9  int dfs(int u, int p = 0) {
10     subtree_size[u] = 1;
11
12     for(int v : adj[u]) {
13         if(v != p && !removed[v]) {
14             subtree_size[u] += dfs(v, u);
15         }
16     }
17
18     return subtree_size[u];
19 }
20
21 int get_centroid(int u, int sz, int p = 0) {
22     for(int v : adj[u]) {
23         if(v != p && !removed[v]) {
24             if(subtree_size[v]*2 > sz) {
25                 return get_centroid(v, sz, u);
26             }
27         }
28     }
29
30     return u;
31 }
32
33 char get_next(char c) {
34     if (c != 'Z') return c + 1;
35     return '$';
36 }
37
38 bool flag = true;
39
40 void solve(int node, char c) {
41     int center = get_centroid(node, dfs(node));
42     ans[center] = c;
43     removed[center] = true;
44
45     for (auto u : adj[center]) {
46         if (!removed[u]) {
47             char next = get_next(c);
48             if (next == '$') {
49                 flag = false;
50                 return;
51             }
52             solve(u, next);
53         }
54     }
55 }
56
57 int32_t main(){
58     ios::sync_with_stdio(false);
59     cin.tie(NULL);
60
61     cin >> n;
62     adj.resize(n + 1);
63     ans.resize(n + 1);
64     removed.resize(n + 1);
65     subtree_size.resize(n + 1);
66
67     for (int i = 1; i <= n - 1; i++) {
68         int u, v; cin >> u >> v;
69         adj[u].insert(v);
70         adj[v].insert(u);
71     }
72
73     solve(1, 'A');
74

```

```

75     if (!flag) cout << "Impossible!\n";
76     else {
77         for (int i = 1; i <= n; i++) {
78             cout << ans[i] << ' ';
79         }
80         cout << '\n';
81     }
82
83     return 0;
84 }

```

6.13 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 }

```

6.14 Small To Large

```

1 // Problem:
2 // https://codeforces.com/contest/600/problem/E
3
4 void process_colors(int curr, int parent) {
5
6     for (int n : adj[curr]) {
7         if (n != parent) {
8             process_colors(n, curr);
9
10             if (colors[curr].size() < colors[n].size
                ()) {

```

```

11         sum_num[curr] = sum_num[n];
12         vmax[curr] = vmax[n];
13         swap(colors[curr], colors[n]);
14     }
15
16     for (auto [item, vzs] : colors[n]) {
17         if (colors[curr][item] + vzs > vmax[curr]) {
18             vmax[curr] = colors[curr][item] + vzs;
19             sum_num[curr] = item;
20         } else if (colors[curr][item] + vzs ==
21             vmax[curr]) {
22             sum_num[curr] += item;
23         }
24
25         colors[curr][item] += vzs;
26     }
27 }
28 }
29 }
30 }
31
32 int32_t main() {
33     int n; cin >> n;
34
35     for (int i = 1; i <= n; i++) {
36         int a; cin >> a;
37         colors[i][a] = 1;
38         vmax[i] = 1;
39         sum_num[i] = a;
40     }
41
42     for (int i = 1; i < n; i++) {
43         int a, b; cin >> a >> b;
44
45         adj[a].push_back(b);
46         adj[b].push_back(a);
47     }
48
49     process_colors(1, 0);
50
51     for (int i = 1; i <= n; i++) {
52         cout << sum_num[i] << (i < n ? " " : "\n");
53     }
54
55     return 0;
56 }
57
58 }
59
60

```

6.15 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 }
32
33 }
34
35 }
36
37 }
38
39 }
40
41 }
42
43 }
44
45 }
46
47 }
48
49 }
50
51 }
52
53 }
54
55 }
56
57 }
58
59 }
60

```

```

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 }
32
33 }
34
35 }
36
37 }
38
39 }
40
41 }
42
43 }
44
45 }
46
47 }
48
49 }
50
51 }
52
53 }
54
55 }
56
57 }
58
59 }
60

```

6.16 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
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

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

```

6.17 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     point operator/(cod t) const{ return {x/t, y/t}; }
22
23     cod operator*(const point &o) const{ return x * o
24         .x + y * o.y; }
25     cod operator^(const point &o) const{ return x * o
26         .y - y * o.x; }
27     bool operator<(const point &o) const{
28         return (eq(x, o.x) ? y < o.y : x < o.x);
29     }
30     bool operator==(const point &o) const{
31         return eq(x, o.x) and eq(y, o.y);
32     }
33     friend ostream& operator<<(ostream& os, point p) {
34         return os << "(" << p.x << ", " << p.y << ")"; }
35 };
36
37 int ccw(point a, point b, point e){ // -1=dir; 0=
38     collinear; 1=esq;
39     cod tmp = (b-a) ^ (e-a); // vector from a to b
40     return (tmp > EPS) - (tmp < -EPS);
41 }
42
43 ld norm(point a){ // Modulo

```

```

38     return sqrt(a * a);
39 }
40 cod norm2(point a){
41     return a * a;
42 }
43 bool nulo(point a){
44     return (eq(a.x, 0) and eq(a.y, 0));
45 }
46 point rotccw(point p, ld a){
47     // a = PI*a/180; // graus
48     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+p.x*sin(a)));
49 }
50 point rot90cw(point a) { return point(a.y, -a.x); };
51 point rot90ccw(point a) { return point(-a.y, a.x); };
52
53 ld proj(point a, point b){ // a sobre b
54     return a*b/norm(b);
55 }
56 ld angle(point a, point b){ // em radianos
57     ld ang = a*b / norm(a) / norm(b);
58     return acos(max(min(ang, (ld)1), (ld)-1));
59 }
60 ld angle_vec(point v){
61     // return 180/PI*atan2(v.x, v.y); // graus
62     return atan2(v.x, v.y);
63 }
64 ld order_angle(point a, point b){ // from a to b ccw
65     (a in front of b)
66     ld aux = angle(a,b)*180/PI;
67     return ((a^b)<=0 ? aux:360-aux);
68 }
69 bool angle_less(point a1, point b1, point a2, point b2){ // ang(a1,b1) <= ang(a2,b2)
70     point p1((a1*b1), abs((a1^b1)));
71     point p2((a2*b2), abs((a2^b2)));
72     return (p1^p2) <= 0;
73 }
74 ld area(vp &p){ // (points sorted)
75     ld ret = 0;
76     for(int i=2;i<(int)p.size();i++)
77         ret += (p[i]-p[0])^(p[i-1]-p[0]);
78     return abs(ret/2);
79 }
80 ld areaT(point &a, point &b, point &c){
81     return abs((b-a)^(c-a))/2.0;
82 }
83
84 point center(vp &A){
85     point c = point();
86     int len = A.size();
87     for(int i=0;i<len;i++)
88         c=c+A[i];
89     return c/len;
90 }
91
92 point forca_mod(point p, ld m){
93     ld cm = norm(p);
94     if(cm<EPS) return point();
95     return point(p.x*m/cm,p.y*m/cm);
96 }
97
98 ld param(point a, point b, point v){
99     // v = t*(b-a) + a // return t;
100    // assert(line(a, b).inside_seg(v));
101    return ((v-a) * (b-a)) / ((b-a) * (b-a));
102 }
103
104 bool simetric(vp &a){ //ordered
105     int n = a.size();
106     point c = center(a);
107     if(n&1) return false;

```

```

108     for(int i=0;i<n/2;i++)
109         if(ccw(a[i], a[i+n/2], c) != 0)
110             return false;
111     return true;
112 }
113
114 point mirror(point m1, point m2, point p){
115     // mirror point p around segment m1m2
116     point seg = m2-m1;
117     ld t0 = ((p-m1)*seg) / (seg*seg);
118     point ort = m1 + seg*t0;
119     point pm = ort-(p-ort);
120     return pm;
121 }
122
123 ///////////////
124 // Line //
125 ///////////////
126
127 struct line{
128     point p1, p2;
129     cod a, b, c; // ax+by+c = 0;
130     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
131     line(point p1=0, point p2=0): p1(p1), p2(p2){
132         a = p1.y - p2.y;
133         b = p2.x - p1.x;
134         c = p1 ^ p2;
135     }
136     line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
137     {
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 -
seg
190     if((p-a)*(b-a) < EPS) return norm(p-a);
191     if((p-b)*(a-b) < EPS) return norm(p-b);
192     return abs((p-a)^(b-a)) / norm(b-a);
193 }
194
195 ld dist_line(point p, line l){ // point - line
196     return abs(l.eval(p))/sqrt(1.a*1.a + 1.b*1.b);
197 }
198
199 line bisector(point a, point b){
200     point d = (b-a)*2;
201     return line(d.x, d.y, a*a - b*b);
202 }
203
204 line perpendicular(line l, point p){ // passes
through p
205     return line(1.b, -1.a, -1.b*p.x + 1.a*p.y);
206 }
207
208 ///////////////
209 // Circle //
210 ///////////////
211
212 struct circle{
213     point c; cod r;
214     circle() : c(0, 0), r(0){}
215     circle(const point o) : c(o), r(0){}
216     circle(const point a, const point b){
217         c = (a+b)/2;
218         r = norm(a-c);
219     }
220
221     circle(const point a, const point b, const point
cc){
222         assert(ccw(a, b, cc) != 0);
223         c = inter_line(bisector(a, b), bisector(b, cc
))[0];
224         r = norm(a-c);
225     }
226     bool inside(const point &a) const{
227         return norm(a - c) <= r + EPS;
228     }
229 };
230
231 pair<point, point> tangent_points(circle cr, point p)
{
232     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
233     point p1 = rotccw(cr.c-p, -theta);
234     point p2 = rotccw(cr.c-p, theta);
235     assert(d1 >= cr.r);
236     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
237     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
238     return {p1, p2};
239 }
240
241
242 circle incircle(point p1, point p2, point p3){
243     ld m1 = norm(p2-p3);
244     ld m2 = norm(p1-p3);
245     ld m3 = norm(p1-p2);
246     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
247     ld s = 0.5*(m1+m2+m3);
248     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
249     return circle(c, r);
250 }
251
252 circle circumcircle(point a, point b, point c) {
253     circle ans;
254     point u = point((b-a).y, -(b-a).x);
255     point v = point((c-a).y, -(c-a).x);
256     point n = (c-b)*0.5;
257     ld t = (u^n)/(v^u);
258     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.
p1)*(ab) / (ab*ab));
265     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
/ (ab*ab);
266     if (h2 < -EPS) return {};
267     if (eq(h2, 0)) return {p};
268     point h = (ab/norm(ab)) * sqrt(h2);
269     return {p - h, p + h};
270 }
271
272 vp inter_circle(circle C1, circle C2){
273     if(C1.c == C2.c) { assert(C1.r != C2.r); return
{}; }
274     point vec = C2.c - C1.c;
275     ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
;
276     ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
C1.r*C1.r - p*p*d2;
277     if (sum*sum < d2 or dif*dif > d2) return {};
278     point mid = C1.c + vec*p, per = point(-vec.y, vec
.x) * sqrt(max((ld)0, h2) / d2);
279     if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
280     return {mid + per, mid - per};
281 }
282
283 // minimum circle cover O(n) amortizado
284 circle min_circle_cover(vp v){
285     random_shuffle(v.begin(), v.end());
286     circle ans;
287     int n = v.size();
288     for(int i=0;i<n;i++) if(!ans.inside(v[i])){
289         ans = circle(v[i]);
290         for(int j=0;j<i;j++) if(!ans.inside(v[j])){
291             ans = circle(v[i], v[j]);
292             for(int k=0;k<j;k++) if(!ans.inside(v[k]))
293                 ans = circle(v[i], v[j], v[k]);
294         }
295     }
296     return ans;
297 }
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

8 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 }

```

```

10 }
11 return lo;
12 }

```

8.5 Ternary Search

```

1 double ternary_search(double l, double r) {
2     double eps = 1e-9; //set the error
3     while (r - l > eps) {
4         double m1 = l + (r - l) / 3;
5         double m2 = r - (r - l) / 3;
6         double f1 = f(m1); //evaluates the
7         double f2 = f(m2); //evaluates the
8         if (f1 < f2)
9             l = m1;
10        else
11            r = m2;
12    }
13    return f(l); //return the
14 }

```

8.6 Binary Search First True

```

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

```

8.7 Biggest K

```

1 // Description: Gets sum of k biggest or k smallest
2 // elements in an array
3 // Problem: https://atcoder.jp/contests/abc306/tasks/abc306\_e
4 // Complexity: O(log n)
5
6 struct SetSum {
7     ll s = 0;
8     multiset<ll> mt;
9     void add(ll x){
10         mt.insert(x);
11         s += x;
12     }
13     int pop(ll x){
14         auto f = mt.find(x);
15         if(f == mt.end()) return 0;
16         mt.erase(f);
17         s -= x;
18         return 1;
19     }
20 }
21 };
22
23 struct BigK {
24     int k;
25     SetSum gt, mt;
26     BigK(int _k){
27         k = _k;
28     }
29 }

```

```

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.
           begin()) {
53             int v = *small.rbegin();
54             small.erase(prev(small.end()));
55             big.insert(v);
56             sums -= v;
57             sumb += v;
58         }
59         balance();
60     }
61
62     bool rem(int x) {
63         n--;
64         auto it1 = small.find(x);
65         auto it2 = big.find(x);
66         bool flag = false;
67         if (it1 != small.end()) {
68             sums -= *it1;
69             small.erase(it1);
70             flag = true;
71         } else if (it2 != big.end()) {
72             sumb -= *it2;
73             big.erase(it2);
74             flag = true;
75         }
76         balance();
77         return flag;
78     }
79

```



```

80 ll sum_small() {
81     return sums;
82 }
83
84 ll sum_big() {
85     return sumb;
86 }
87
88 int median() {
89     return *big.begin();
90 }
91 };

```

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     }

```

```

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     void update(int pos, int ini, int fim, int id,
59         int val) {
60         if (ini > id || fim < id) {
61             return;
62         }
63
64         if (ini == fim) {
65             seg[pos] = val;
66
67             return;
68         }
69
70         int m = (ini + fim) >> 1;
71
72         if (id <= m) {
73             if (e[pos] == 0) e[pos] = create();
74             update(e[pos], ini, m, id, val);
75         } else {
76             if (d[pos] == 0) d[pos] = create();
77             update(d[pos], m + 1, fim, id, val);
78         }
79
80         seg[pos] = f(seg[e[pos]], seg[d[pos]]);
81     }
82
83     ftype query(int p, int q) {
84         return query(1, 1, n, p, q);
85     }
86
87     void update(int id, int val) {
88         update(1, 1, n, id, val);
89     }
90 };

```

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
10 // Problem:
11 // https://cses.fi/problemset/task/1739/
12
13 // Complexity:
14 // Time complexity:
15 // O(log N * log M) for both query and update
16 // O(N * M) for build
17 // Memory complexity:
18 // 4 * M * N
19
20 // How to use:
21 // Segtree2D seg = Segtree2D(n, n);
22 // vector<vector<int>>> v(n, vector<int>(n));
23 // seg.build(v);
24
25 // Notes
26 // Indexed at zero
27
28 struct Segtree2D {
29     const int MAXN = 1025;
30     int N, M;
31
32     void build(const vector<vector<int>>& v) {
33         N = v.size();
34         M = v[0].size();
35         for (int i = 0; i < N; i++)
36             for (int j = 0; j < M; j++)
37                 update(i, j, v[i][j]);
38     }
39
40     int query(int i1, int j1, int i2, int j2) {
41         return query(i1, j1, i2, j2, 0, 0);
42     }
43
44     void update(int i, int j, int val) {
45         update(i, j, val, 0, 0);
46     }
47
48     void update(int i, int j, int val, int ni, int nj) {
49         if (i < 0 || i >= N || j < 0 || j >= M) return;
50         if (ni < 0 || ni >= N || nj < 0 || nj >= M) return;
51         if (i == ni && j == nj) {
52             seg[i][j] = val;
53             return;
54         }
55         int mi = (ni + nj) >> 1;
56         if (i <= mi)
57             update(i, j, val, ni, mi);
58         else
59             update(i, j, val, mi, nj);
60         seg[i][j] = f(seg[i][mi], seg[mi][j]);
61     }
62
63     ftype f(ftype a, ftype b) {
64         return a + b;
65     }
66 };

```

```

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];
43             }else{
44                 seg[noX][noY] = seg[2*noX+1][noY] +
45                 seg[2*noX+2][noY];
46             }
47         }else{
48             int m = (lY+rY)/2;
49
50             buildY(noX, lX, rX, 2*noY+1, lY, m, v);
51             buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);
52
53             seg[noX][noY] = seg[noX][2*noY+1] + seg[
54             noX][2*noY+2];
55         }
56     }
57
58     void buildX(int noX, int lX, int rX, vector<
59     vector<int>>> &v){
60         if(lX != rX){
61             int m = (lX+rX)/2;
62
63             buildX(2*noX+1, lX, m, v);
64             buildX(2*noX+2, m+1, rX, v);
65         }
66
67         buildY(noX, lX, rX, 0, 0, M - 1, v);
68     }
69
70     void updateY(int noX, int lX, int rX, int noY,
71     int lY, int rY, int y){
72         if(lY == rY){
73             if(lX == rX){
74                 seg[noX][noY] = !seg[noX][noY];
75             }else{
76                 seg[noX][noY] = seg[2*noX+1][noY] +
77                 seg[2*noX+2][noY];
78             }
79         }else{
80             int m = (lY+rY)/2;
81
82             if(y <= m){
83                 updateY(noX, lX, rX, 2*noY+1, lY, m, y
84             );
85             }else if(m < y){
86                 updateY(noX, lX, rX, 2*noY+2, m+1, rY
87             , y);
88             }
89
90             seg[noX][noY] = seg[noX][2*noY+1] + seg[
91             noX][2*noY+2];
92         }
93     }
94
95     void updateX(int noX, int lX, int rX, int x, int
96     y){
97         int m = (lX+rX)/2;
98
99         if(lX != rX){
100             if(x <= m){
101                 updateX(2*noX+1, lX, m, x, y);
102             }else if(m < x){
103                 updateX(2*noX+2, m+1, rX, x, y);
104             }
105         }
106
107         updateY(noX, lX, rX, 0, 0, M - 1, y);
108     }
109
110     int queryY(int noX, int noY, int lY, int rY, int
111     aY, int bY){
112         if(aY <= lY && rY <= bY) return seg[noX][noY
113     ];
114
115         int m = (lY+rY)/2;
116
117         if(bY <= m) return queryY(noX, 2*noY+1, lY, m
118     , aY, bY);
119         if(m < aY) return queryY(noX, 2*noY+2, m+1,
120     rY, aY, bY);
121
122         return queryY(noX, 2*noY+1, lY, m, aY, bY) +
123         queryY(noX, 2*noY+2, m+1, rY, aY, bY);
124     }
125
126     int queryX(int noX, int lX, int rX, int aX, int
127     bX, int aY, int bY){
128         if(aX <= lX && rX <= bX) return queryY(noX,
129     0, 0, M - 1, aY, bY);
130
131         int m = (lX+rX)/2;
132
133         if(bX <= m) return queryX(2*noX+1, lX, m, aX,
134     bX, aY, bY);
135         if(m < aX) return queryX(2*noX+2, m+1, rX, aX
136     , bX, aY, bY);
137
138         return queryX(2*noX+1, lX, m, aX, bX, aY, bY)
139     + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
140     }
141
142     void build(vector<vector<int>>> &v) {
143         buildX(0, 0, N - 1, v);
144     }
145
146     int query(int aX, int bX, int aY, int bY) {
147         return queryX(0, 0, N - 1, aX, bX, aY, bY);
148     }
149
150     void update(int x, int y) {
151         updateX(0, 0, N - 1, x, y);
152     }
153 };

```

9.7 Minimum And Amount

```

1 // Description:
2 // Query - get minimum element in a range (l, r)
3 // inclusive
4 // and also the number of times it appears in that
5 // range
6 // Update - update element at position id to a value
7 // val
8
9 // Problem:
10 // https://codeforces.com/edu/course/2/lesson/4/1/
11 // practice/contest/273169/problem/C
12
13 // Complexity:
14 // O(log n) for both query and update
15
16 // How to use:
17 // Segtree seg = Segtree(n);
18 // seg.build(v);

```

```

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);
29
30     Segtree(int n) {
31         int sz = 1;
32         while (sz < n) sz *= 2;
33         this->n = sz;
34
35         seg.assign(2*sz, NEUTRAL);
36     }
37
38     ftype f(ftype a, ftype b) {
39         if (a.ff < b.ff) return a;
40         if (b.ff < a.ff) return b;
41
42         return mp(a.ff, a.ss + b.ss);
43     }
44
45     ftype query(int pos, int ini, int fim, int p, int q) {
46         if (ini >= p && fim <= q) {
47             return seg[pos];
48         }
49
50         if (q < ini || p > fim) {
51             return NEUTRAL;
52         }
53
54         int e = 2*pos + 1;
55         int d = 2*pos + 2;
56         int m = ini + (fim - ini) / 2;
57
58         return f(query(e, ini, m, p, q), query(d, m +
59 1, fim, p, q));
60
61     void update(int pos, int ini, int fim, int id,
62 int val) {
63         if (ini > id || fim < id) {
64             return;
65         }
66
67         if (ini == id && fim == id) {
68             seg[pos] = mp(val, 1);
69
70             return;
71         }
72
73         int e = 2*pos + 1;
74         int d = 2*pos + 2;
75         int m = ini + (fim - ini) / 2;
76
77         update(e, ini, m, id, val);
78         update(d, m + 1, fim, id, val);
79
80         seg[pos] = f(seg[e], seg[d]);
81     }
82
83     void build(int pos, int ini, int fim, vector<int>
84 &v) {
85         if (ini == fim) {
86             if (ini < (int)v.size()) {

```

```

85         seg[pos] = mp(v[ini], 1);
86     }
87     return;
88 }
89
90 int e = 2*pos + 1;
91 int d = 2*pos + 2;
92 int m = ini + (fim - ini) / 2;
93
94 build(e, ini, m, v);
95 build(d, m + 1, fim, v);
96
97 seg[pos] = f(seg[e], seg[d]);
98 }
99
100 ftype query(int p, int q) {
101     return query(0, 0, n - 1, p, q);
102 }
103
104 void update(int id, int val) {
105     update(0, 0, n - 1, id, val);
106 }
107
108 void build(vector<int> &v) {
109     build(0, 0, n - 1, v);
110 }
111
112 void debug() {
113     for (auto e : seg) {
114         cout << e.ff << ' ' << e.ss << '\n';
115     }
116     cout << '\n';
117 }
118 };

```

9.8 Lazy Addition To Segment

```

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

```

```

33         seg.assign(2*sz, NEUTRAL);
34         lazy.assign(2*sz, NEUTRAL_LAZY);
35     }
36
37     ftype apply_lazy(ftype a, ftype b, int len) {
38         if (b == NEUTRAL_LAZY) return a;
39         if (a == NEUTRAL_LAZY) return b * len;
40         else return a + b * len;
41     }
42
43     void propagate(int pos, int ini, int fim) {
44         if (ini == fim) {
45             return;
46         }
47
48         int e = 2*pos + 1;
49         int d = 2*pos + 2;
50         int m = ini + (fim - ini) / 2;
51
52         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
53         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
54
55         seg[e] = apply_lazy(seg[e], lazy[pos], m -
56 ini + 1);
57         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
58 m);
59
60         lazy[pos] = NEUTRAL_LAZY;
61     }
62
63     ftype f(ftype a, ftype b) {
64         return a + b;
65     }
66
67     ftype query(int pos, int ini, int fim, int p, int
68 q) {
69         propagate(pos, ini, fim);
70
71         if (ini >= p && fim <= q) {
72             return seg[pos];
73         }
74
75         if (q < ini || p > fim) {
76             return NEUTRAL;
77         }
78
79         int e = 2*pos + 1;
80         int d = 2*pos + 2;
81         int m = ini + (fim - ini) / 2;
82
83         return f(query(e, ini, m, p, q), query(d, m +
84 1, fim, p, q));
85     }
86
87     void update(int pos, int ini, int fim, int p, int
88 q, int val) {
89         propagate(pos, ini, fim);
90
91         if (ini > q || fim < p) {
92             return;
93         }
94
95         if (ini >= p && fim <= q) {
96             lazy[pos] = apply_lazy(lazy[pos], val, 1)
97 ;
98             seg[pos] = apply_lazy(seg[pos], val, fim
99 - ini + 1);
100
101             return;
102         }
103
104         int e = 2*pos + 1;
105         int d = 2*pos + 2;
106         int m = ini + (fim - ini) / 2;
107
108         build(e, ini, m, v);
109         build(d, m + 1, fim, v);
110
111         seg[pos] = f(seg[e], seg[d]);
112     }
113
114     ftype query(int p, int q) {
115         return query(0, 0, n - 1, p, q);
116     }
117
118     void update(int p, int q, int val) {
119         update(0, 0, n - 1, p, q, val);
120     }
121
122     void build(vector<int> &v) {
123         build(0, 0, n - 1, v);
124     }
125
126     void debug() {
127         for (auto e : seg) {
128             cout << e << ' ';
129         }
130         cout << '\n';
131         for (auto e : lazy) {
132             cout << e << ' ';
133         }
134         cout << '\n';
135         cout << '\n';
136     }
137
138     int d = 2*pos + 2;
139     int m = ini + (fim - ini) / 2;
140
141     update(e, ini, m, p, q, val);
142     update(d, m + 1, fim, p, q, val);
143
144     seg[pos] = f(seg[e], seg[d]);
145 }
146
147 void build(int pos, int ini, int fim, vector<int>
148 &v) {
149     if (ini == fim) {
150         if (ini < (int)v.size()) {
151             seg[pos] = v[ini];
152         }
153         return;
154     }
155
156     int e = 2*pos + 1;
157     int d = 2*pos + 2;
158     int m = ini + (fim - ini) / 2;
159
160     build(e, ini, m, v);
161     build(d, m + 1, fim, v);
162
163     seg[pos] = f(seg[e], seg[d]);
164 }
165
166 ftype query(int p, int q) {
167     return query(0, 0, n - 1, p, q);
168 }
169
170 void update(int p, int q, int val) {
171     update(0, 0, n - 1, p, q, val);
172 }
173
174 void build(vector<int> &v) {
175     build(0, 0, n - 1, v);
176 }
177
178 void debug() {
179     for (auto e : seg) {
180         cout << e << ' ';
181     }
182     cout << '\n';
183     for (auto e : lazy) {
184         cout << e << ' ';
185     }
186     cout << '\n';
187     cout << '\n';
188 }
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

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
18 // Notes
19 // The maximum segment sum can be a negative number
20 // In that case, taking zero elements is the best
  choice
21 // So we need to take the maximum between 0 and the
  query
22 // max(OLL, seg.query(0, n).max_seg)
23
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
40 typedef Node ftype;
41
42 struct Segtree {
43     vector<ftype> seg;
44     int n;
45     const ftype NEUTRAL = Node(0, 0, 0, 0);
46
47     Segtree(int n) {
48         int sz = 1;
49         // potencia de dois mais proxima
50         while (sz < n) sz *= 2;
51         this->n = sz;
52
53         // numero de nos da seg
54         seg.assign(2*sz, NEUTRAL);
55     }
56
57     ftype f(ftype a, ftype b) {
58         ftype_node max_seg = max({a.max_seg, b.
59         max_seg, a.suf + b.pref});
60         ftype_node pref = max(a.pref, a.sum + b.pref);
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
88         if (ini > id || fim < id) {
89             return;
90         }
91
92         if (ini == id && fim == id) {
93             seg[pos] = Node(val, val, val, val);
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, id, val);
103         update(d, m + 1, fim, id, val);
104
105         seg[pos] = f(seg[e], seg[d]);
106     }
107
108     void build(int pos, int ini, int fim, vector<int>
109     &v) {
110         if (ini == fim) {
111             // se a posição existir no array original
112             // seg tamanho potencia de dois
113             if (ini < (int)v.size()) {
114                 seg[pos] = Node(v[ini], v[ini], v[ini]
115                 ], v[ini]);
116             }
117             return;
118         }
119
120         int e = 2*pos + 1;
121         int d = 2*pos + 2;
122         int m = ini + (fim - ini) / 2;
123
124         build(e, ini, m, v);
125         build(d, m + 1, fim, v);
126
127         seg[pos] = f(seg[e], seg[d]);
128     }
129
130     ftype query(int p, int q) {
131         return query(0, 0, n - 1, p, q);
132     }
133
134     void update(int id, int val) {
135         update(0, 0, n - 1, id, val);
136     }
137
138     void build(vector<int> &v) {
139         build(0, 0, n - 1, v);
140     }
141
142     void debug() {
143         for (auto e : seg) {
144             cout << e.max_seg << ' ' << e.pref << ' '
145             << e.suf << ' ' << e.sum << '\n';
146         }
147         cout << '\n';
148     }
149 };

```

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
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(1, 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 +
  1, fim, p, q));
57     }
58
59     void update(int pos, int ini, int fim, int id,
  int val) {
60         if (ini > id || fim < id) {
61             return;
62         }
63
64         if (ini == id && fim == id) {
65             seg[pos] = val;
66
67             return;
68         }
69
70         int e = 2*pos + 1;
71         int d = 2*pos + 2;
72         int m = ini + (fim - ini) / 2;
73
74         update(e, ini, m, id, val);
75         update(d, m + 1, fim, id, val);
76
77         seg[pos] = f(seg[e], seg[d]);
78     }
79
80     void build(int pos, int ini, int fim, vector<int>
  &v) {
81         if (ini == fim) {
82             if (ini < (int)v.size()) {
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 = -INF;
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 -
42 ini + 1);
43     seg[d] = apply_lazy(seg[d], lazy[pos], fim -
44 m);
45
46     lazy[pos] = NEUTRAL_LAZY;
47 }
48
49 ftype f(ftype a, ftype b) {
50     return a + b;
51 }
52
53 ftype query(int pos, int ini, int fim, int p, int q) {
54     propagate(pos, ini, fim);
55
56     if (ini >= p && fim <= q) {
57         return seg[pos];
58     }
59
60     if (q < ini || p > fim) {
61         return NEUTRAL;
62     }
63
64     int e = 2*pos + 1;
65     int d = 2*pos + 2;
66     int m = ini + (fim - ini) / 2;
67
68     return f(query(e, ini, m, p, q), query(d, m
69 + 1, fim, p, q));
70 }
71
72 void update(int pos, int ini, int fim, int p, int q, int val) {
73     propagate(pos, ini, fim);
74
75     if (ini > q || fim < p) {
76         return;
77     }
78
79     if (ini >= p && fim <= q) {
80         lazy[pos] = apply_lazy(lazy[pos], val, 1);
81     }
82
83     seg[pos] = apply_lazy(seg[pos], val, fim
84 - ini + 1);
85
86     return;
87 }
88
89 int e = 2*pos + 1;
90 int d = 2*pos + 2;
91 int m = ini + (fim - ini) / 2;
92
93 update(e, ini, m, p, q, val);
94 update(d, m + 1, fim, p, q, val);
95
96 seg[pos] = f(seg[e], seg[d]);
97 }
98
99 else return b * len;
100 }
101
102 void propagate(int pos, int ini, int fim) {
103     if (ini == fim) {
104         return;
105     }
106
107     int e = 2*pos + 1;
108     int d = 2*pos + 2;
109     int m = ini + (fim - ini) / 2;
110
111     lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
112     lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
113
114     seg[e] = apply_lazy(seg[e], lazy[pos], m -
115 ini + 1);
116     seg[d] = apply_lazy(seg[d], lazy[pos], fim -
117 m);
118
119     lazy[pos] = NEUTRAL_LAZY;
120 }
121
122 ftype f(ftype a, ftype b) {
123     return a + b;
124 }
125
126 ftype query(int pos, int ini, int fim, int p, int q) {
127     propagate(pos, ini, fim);
128
129     if (ini >= p && fim <= q) {
130         return seg[pos];
131     }
132
133     if (q < ini || p > fim) {
134         return NEUTRAL;
135     }
136
137     int e = 2*pos + 1;
138     int d = 2*pos + 2;
139     int m = ini + (fim - ini) / 2;
140
141     return f(query(e, ini, m, p, q), query(d, m
142 + 1, fim, p, q));
143 }
144
145 void update(int pos, int ini, int fim, int p, int q, int val) {
146     propagate(pos, ini, fim);
147
148     if (ini > q || fim < p) {
149         return;
150     }
151
152     if (ini >= p && fim <= q) {
153         lazy[pos] = apply_lazy(lazy[pos], val, 1);
154     }
155
156     seg[pos] = apply_lazy(seg[pos], val, fim
157 - ini + 1);
158
159     return;
160 }
161
162 int e = 2*pos + 1;
163 int d = 2*pos + 2;
164 int m = ini + (fim - ini) / 2;
165
166 update(e, ini, m, p, q, val);
167 update(d, m + 1, fim, p, q, val);
168
169 seg[pos] = f(seg[e], seg[d]);
170 }
171
172 void build(int pos, int ini, int fim, vector<int>
173 &v) {
174     if (ini == fim) {
175         // se a posição existir no array original
176         // seg tamanho potencia de dois
177         if (ini < (int)v.size()) {
178             seg[pos] = v[ini];
179         }
180         return;
181     }
182
183     int e = 2*pos + 1;
184     int d = 2*pos + 2;
185     int m = ini + (fim - ini) / 2;
186
187     build(e, ini, m, v);
188     build(d, m + 1, fim, v);
189
190     seg[pos] = f(seg[e], seg[d]);
191 }
192
193 ftype query(int p, int q) {
194     return query(0, 0, n - 1, p, q);
195 }
196
197 void update(int p, int q, int val) {
198     update(0, 0, n - 1, p, q, val);
199 }
200
201 void build(vector<int> &v) {
202     build(0, 0, n - 1, v);
203 }
204
205 void debug() {
206     for (auto e : seg) {
207         cout << e << ' ';
208     }
209     cout << '\n';
210     for (auto e : lazy) {
211         cout << e << ' ';
212     }
213     cout << '\n';
214     cout << '\n';
215 }
216
217 void build(int pos, int ini, int fim, vector<int>
218 &v) {
219     if (ini == fim) {
220         // se a posição existir no array original
221         // seg tamanho potencia de dois
222         if (ini < (int)v.size()) {
223             seg[pos] = v[ini];
224         }
225         return;
226     }
227
228     int e = 2*pos + 1;
229     int d = 2*pos + 2;
230     int m = ini + (fim - ini) / 2;
231
232     build(e, ini, m, v);
233     build(d, m + 1, fim, v);
234
235     seg[pos] = f(seg[e], seg[d]);
236 }
237
238 ftype query(int p, int q) {
239     return query(0, 0, n - 1, p, q);
240 }
241
242 void update(int p, int q, int val) {
243     update(0, 0, n - 1, p, q, val);
244 }
245
246 void build(vector<int> &v) {
247     build(0, 0, n - 1, v);
248 }
249
250 void debug() {
251     for (auto e : seg) {
252         cout << e << ' ';
253     }
254     cout << '\n';
255     for (auto e : lazy) {
256         cout << e << ' ';
257     }
258     cout << '\n';
259     cout << '\n';
260 }
261
262 void build(int pos, int ini, int fim, vector<int>
263 &v) {
264     if (ini == fim) {
265         // se a posição existir no array original
266         // seg tamanho potencia de dois
267         if (ini < (int)v.size()) {
268             seg[pos] = v[ini];
269         }
270         return;
271     }
272
273     int e = 2*pos + 1;
274     int d = 2*pos + 2;
275     int m = ini + (fim - ini) / 2;
276
277     build(e, ini, m, v);
278     build(d, m + 1, fim, v);
279
280     seg[pos] = f(seg[e], seg[d]);
281 }
282
283 ftype query(int p, int q) {
284     return query(0, 0, n - 1, p, q);
285 }
286
287 void update(int p, int q, int val) {
288     update(0, 0, n - 1, p, q, val);
289 }
290
291 void build(vector<int> &v) {
292     build(0, 0, n - 1, v);
293 }
294
295 void debug() {
296     for (auto e : seg) {
297         cout << e << ' ';
298     }
299     cout << '\n';
300     for (auto e : lazy) {
301         cout << e << ' ';
302     }
303     cout << '\n';
304     cout << '\n';
305 }
306
307 void build(int pos, int ini, int fim, vector<int>
308 &v) {
309     if (ini == fim) {
310         // se a posição existir no array original
311         // seg tamanho potencia de dois
312         if (ini < (int)v.size()) {
313             seg[pos] = v[ini];
314         }
315         return;
316     }
317
318     int e = 2*pos + 1;
319     int d = 2*pos + 2;
320     int m = ini + (fim - ini) / 2;
321
322     build(e, ini, m, v);
323     build(d, m + 1, fim, v);
324
325     seg[pos] = f(seg[e], seg[d]);
326 }
327
328 ftype query(int p, int q) {
329     return query(0, 0, n - 1, p, q);
330 }
331
332 void update(int p, int q, int val) {
333     update(0, 0, n - 1, p, q, val);
334 }
335
336 void build(vector<int> &v) {
337     build(0, 0, n - 1, v);
338 }
339
340 void debug() {
341     for (auto e : seg) {
342         cout << e << ' ';
343     }
344     cout << '\n';
345     for (auto e : lazy) {
346         cout << e << ' ';
347     }
348     cout << '\n';
349     cout << '\n';
350 }
351
352 void build(int pos, int ini, int fim, vector<int>
353 &v) {
354     if (ini == fim) {
355         // se a posição existir no array original
356         // seg tamanho potencia de dois
357         if (ini < (int)v.size()) {
358             seg[pos] = v[ini];
359         }
360         return;
361     }
362
363     int e = 2*pos + 1;
364     int d = 2*pos + 2;
365     int m = ini + (fim - ini) / 2;
366
367     build(e, ini, m, v);
368     build(d, m + 1, fim, v);
369
370     seg[pos] = f(seg[e], seg[d]);
371 }
372
373 ftype query(int p, int q) {
374     return query(0, 0, n - 1, p, q);
375 }
376
377 void update(int p, int q, int val) {
378     update(0, 0, n - 1, p, q, val);
379 }
380
381 void build(vector<int> &v) {
382     build(0, 0, n - 1, v);
383 }
384
385 void debug() {
386     for (auto e : seg) {
387         cout << e << ' ';
388     }
389     cout << '\n';
390     for (auto e : lazy) {
391         cout << e << ' ';
392     }
393     cout << '\n';
394     cout << '\n';
395 }
396
397 void build(int pos, int ini, int fim, vector<int>
398 &v) {
399     if (ini == fim) {
400         // se a posição existir no array original
401         // seg tamanho potencia de dois
402         if (ini < (int)v.size()) {
403             seg[pos] = v[ini];
404         }
405         return;
406     }
407
408     int e = 2*pos + 1;
409     int d = 2*pos + 2;
410     int m = ini + (fim - ini) / 2;
411
412     build(e, ini, m, v);
413     build(d, m + 1, fim, v);
414
415     seg[pos] = f(seg[e], seg[d]);
416 }
417
418 ftype query(int p, int q) {
419     return query(0, 0, n - 1, p, q);
420 }
421
422 void update(int p, int q, int val) {
423     update(0, 0, n - 1, p, q, val);
424 }
425
426 void build(vector<int> &v) {
427     build(0, 0, n - 1, v);
428 }
429
430 void debug() {
431     for (auto e : seg) {
432         cout << e << ' ';
433     }
434     cout << '\n';
435     for (auto e : lazy) {
436         cout << e << ' ';
437     }
438     cout << '\n';
439     cout << '\n';
440 }
441
442 void build(int pos, int ini, int fim, vector<int>
443 &v) {
444     if (ini == fim) {
445         // se a posição existir no array original
446         // seg tamanho potencia de dois
447         if (ini < (int)v.size()) {
448             seg[pos] = v[ini];
449         }
450         return;
451     }
452
453     int e = 2*pos + 1;
454     int d = 2*pos + 2;
455     int m = ini + (fim - ini) / 2;
456
457     build(e, ini, m, v);
458     build(d, m + 1, fim, v);
459
460     seg[pos] = f(seg[e], seg[d]);
461 }
462
463 ftype query(int p, int q) {
464     return query(0, 0, n - 1, p, q);
465 }
466
467 void update(int p, int q, int val) {
468     update(0, 0, n - 1, p, q, val);
469 }
470
471 void build(vector<int> &v) {
472     build(0, 0, n - 1, v);
473 }
474
475 void debug() {
476     for (auto e : seg) {
477         cout << e << ' ';
478     }
479     cout << '\n';
480     for (auto e : lazy) {
481         cout << e << ' ';
482     }
483     cout << '\n';
484     cout << '\n';
485 }
486
487 void build(int pos, int ini, int fim, vector<int>
488 &v) {
489     if (ini == fim) {
490         // se a posição existir no array original
491         // seg tamanho potencia de dois
492         if (ini < (int)v.size()) {
493             seg[pos] = v[ini];
494         }
495         return;
496     }
497
498     int e = 2*pos + 1;
499     int d = 2*pos + 2;
500     int m = ini + (fim - ini) / 2;
501
502     build(e, ini, m, v);
503     build(d, m + 1, fim, v);
504
505     seg[pos] = f(seg[e], seg[d]);
506 }
507
508 ftype query(int p, int q) {
509     return query(0, 0, n - 1, p, q);
510 }
511
512 void update(int p, int q, int val) {
513     update(0, 0, n - 1, p, q, val);
514 }
515
516 void build(vector<int> &v) {
517     build(0, 0, n - 1, v);
518 }
519
520 void debug() {
521     for (auto e : seg) {
522         cout << e << ' ';
523     }
524     cout << '\n';
525     for (auto e : lazy) {
526         cout << e << ' ';
527     }
528     cout << '\n';
529     cout << '\n';
530 }
531
532 void build(int pos, int ini, int fim, vector<int>
533 &v) {
534     if (ini == fim) {
535         // se a posição existir no array original
536         // seg tamanho potencia de dois
537         if (ini < (int)v.size()) {
538             seg[pos] = v[ini];
539         }
540         return;
541     }
542
543     int e = 2*pos + 1;
544     int d = 2*pos + 2;
545     int m = ini + (fim - ini) / 2;
546
547     build(e, ini, m, v);
548     build(d, m + 1, fim, v);
549
550     seg[pos] = f(seg[e], seg[d]);
551 }
552
553 ftype query(int p, int q) {
554     return query(0, 0, n - 1, p, q);
555 }
556
557 void update(int p, int q, int val) {
558     update(0, 0, n - 1, p, q, val);
559 }
560
561 void build(vector<int> &v) {
562     build(0, 0, n - 1, v);
563 }
564
565 void debug() {
566     for (auto e : seg) {
567         cout << e << ' ';
568     }
569     cout << '\n';
570     for (auto e : lazy) {
571         cout << e << ' ';
572     }
573     cout << '\n';
574     cout << '\n';
575 }
576
577 void build(int pos, int ini, int fim, vector<int>
578 &v) {
579     if (ini == fim) {
580         // se a posição existir no array original
581         // seg tamanho potencia de dois
582         if (ini < (int)v.size()) {
583             seg[pos] = v[ini];
584         }
585         return;
586     }
587
588     int e = 2*pos + 1;
589     int d = 2*pos + 2;
590     int m = ini + (fim - ini) / 2;
591
592     build(e, ini, m, v);
593     build(d, m + 1, fim, v);
594
595     seg[pos] = f(seg[e], seg[d]);
596 }
597
598 ftype query(int p, int q) {
599     return query(0, 0, n - 1, p, q);
600 }
601
602 void update(int p, int q, int val) {
603     update(0, 0, n - 1, p, q, val);
604 }
605
606 void build(vector<int> &v) {
607     build(0, 0, n - 1, v);
608 }
609
610 void debug() {
611     for (auto e : seg) {
612         cout << e << ' ';
613     }
614     cout << '\n';
615     for (auto e : lazy) {
616         cout << e << ' ';
617     }
618     cout << '\n';
619     cout << '\n';
620 }
621
622 void build(int pos, int ini, int fim, vector<int>
623 &v) {
624     if (ini == fim) {
625         // se a posição existir no array original
626         // seg tamanho potencia de dois
627         if (ini < (int)v.size()) {
628             seg[pos] = v[ini];
629         }
630         return;
631     }
632
633     int e = 2*pos + 1;
634     int d = 2*pos + 2;
635     int m = ini + (fim - ini) / 2;
636
637     build(e, ini, m, v);
638     build(d, m + 1, fim, v);
639
640     seg[pos] = f(seg[e], seg[d]);
641 }
642
643 ftype query(int p, int q) {
644     return query(0, 0, n - 1, p, q);
645 }
646
647 void update(int p, int q, int val) {
648     update(0, 0, n - 1, p, q, val);
649 }
650
651 void build(vector<int> &v) {
652     build(0, 0, n - 1, v);
653 }
654
655 void debug() {
656     for (auto e : seg) {
657         cout << e << ' ';
658     }
659     cout << '\n';
660     for (auto e : lazy) {
661         cout << e << ' ';
662     }
663     cout << '\n';
664     cout << '\n';
665 }
666
667 void build(int pos, int ini, int fim, vector<int>
668 &v) {
669     if (ini == fim) {
670         // se a posição existir no array original
671         // seg tamanho potencia de dois
672         if (ini < (int)v.size()) {
673             seg[pos] = v[ini];
674         }
675         return;
676     }
677
678     int e = 2*pos + 1;
679     int d = 2*pos + 2;
680     int m = ini + (fim - ini) / 2;
681
682     build(e, ini, m, v);
683     build(d, m + 1, fim, v);
684
685     seg[pos] = f(seg[e], seg[d]);
686 }
687
688 ftype query(int p, int q) {
689     return query(0, 0, n - 1, p, q);
690 }
691
692 void update(int p, int q, int val) {
693     update(0, 0, n - 1, p, q, val);
694 }
695
696 void build(vector<int> &v) {
697     build(0, 0, n - 1, v);
698 }
699
700 void debug() {
701     for (auto e : seg) {
702         cout << e << ' ';
703     }
704     cout << '\n';
705     for (auto e : lazy) {
706         cout << e << ' ';
707     }
708     cout << '\n';
709     cout << '\n';
710 }
711
712 void build(int pos, int ini, int fim, vector<int>
713 &v) {
714     if (ini == fim) {
715         // se a posição existir no array original
716         // seg tamanho potencia de dois
717         if (ini < (int)v.size()) {
718             seg[pos] = v[ini];
719         }
720         return;
721     }
722
723     int e = 2*pos + 1;
724     int d = 2*pos + 2;
725     int m = ini + (fim - ini) / 2;
726
727     build(e, ini, m, v);
728     build(d, m + 1, fim, v);
729
730     seg[pos] = f(seg[e], seg[d]);
731 }
732
733 ftype query(int p, int q) {
734     return query(0, 0, n - 1, p, q);
735 }
736
737 void update(int p, int q, int val) {
738     update(0, 0, n - 1, p, q, val);
739 }
740
741 void build(vector<int> &v) {
742     build(0, 0, n - 1, v);
743 }
744
745 void debug() {
746     for (auto e : seg) {
747         cout << e << ' ';
748     }
749     cout << '\n';
750     for (auto e : lazy) {
751         cout << e << ' ';
752     }
753     cout << '\n';
754     cout << '\n';
755 }
756
757 void build(int pos, int ini, int fim, vector<int>
758 &v) {
759     if (ini == fim) {
760         // se a posição existir no array original
761         // seg tamanho potencia de dois
762         if (ini < (int)v.size()) {
763             seg[pos] = v[ini];
764         }
765         return;
766     }
767
768     int e = 2*pos + 1;
769     int d = 2*pos + 2;
770     int m = ini + (fim - ini) / 2;
771
772     build(e, ini, m, v);
773     build(d, m + 1, fim, v);
774
775     seg[pos] = f(seg[e], seg[d]);
776 }
777
778 ftype query(int p, int q) {
779     return query(0, 0, n - 1, p, q);
780 }
781
782 void update(int p, int q, int val) {
783     update(0, 0, n - 1, p, q, val);
784 }
785
786 void build(vector<int> &v) {
787     build(0, 0, n - 1, v);
788 }
789
790 void debug() {
791     for (auto e : seg) {
792         cout << e << ' ';
793     }
794     cout << '\n';
795     for (auto e : lazy) {
796         cout << e << ' ';
797     }
798     cout << '\n';
799     cout << '\n';
800 }
801
802 void build(int pos, int ini, int fim, vector<int>
803 &v) {
804     if (ini == fim) {
805         // se a posição existir no array original
806         // seg tamanho potencia de dois
807         if (ini < (int)v.size()) {
808             seg[pos] = v[ini];
809         }
810         return;
811     }
812
813     int e = 2*pos + 1;
814     int d = 2*pos + 2;
815     int m = ini + (fim - ini) / 2;
816
817     build(e, ini, m, v);
818     build(d, m + 1, fim, v);
819
820     seg[pos] = f(seg[e], seg[d]);
821 }
822
823 ftype query(int p, int q) {
824     return query(0, 0, n - 1, p, q);
825 }
826
827 void update(int p, int q, int val) {
828     update(0, 0, n - 1, p, q, val);
829 }
830
831 void build(vector<int> &v) {
832     build(0, 0, n - 1, v);
833 }
834
835 void debug() {
836     for (auto e : seg) {
837         cout << e << ' ';
838     }
839     cout << '\n';
840     for (auto e : lazy) {
841         cout << e << ' ';
842     }
843     cout << '\n';
844     cout << '\n';
845 }
846
847 void build(int pos, int ini, int fim, vector<int>
848 &v) {
849     if (ini == fim) {
850         // se a posição existir no array original
851         // seg tamanho potencia de dois
852         if (ini < (int)v.size()) {
853             seg[pos] = v[ini];
854         }
855         return;
856     }
857
858     int e = 2*pos + 1;
859     int d = 2*pos + 2;
860     int m = ini + (fim - ini) / 2;
861
862     build(e, ini, m, v);
863     build(d, m + 1, fim, v);
864
865     seg[pos] = f(seg[e], seg[d]);
866 }
867
868 ftype query(int p, int q) {
869     return query(0, 0, n - 1, p, q);
870 }
871
872 void update(int p, int q, int val) {
873     update(0, 0, n - 1, p, q, val);
874 }
875
876 void build(vector<int> &v) {
877     build(0, 0, n - 1, v);
878 }
879
880 void debug() {
881     for (auto e : seg) {
882         cout << e << ' ';
883     }
884     cout << '\n';
885     for (auto e : lazy) {
886         cout << e << ' ';
887     }
888     cout << '\n';
889     cout << '\n';
890 }
891
892 void build(int pos, int ini, int fim, vector<int>
893 &v) {
894     if (ini == fim) {
895         // se a posição existir no array original
896         // seg tamanho potencia de dois
897         if (ini < (int)v.size()) {
898             seg[pos] = v[ini];
899         }
900         return;
901     }
902
903     int e = 2*pos + 1;
904     int d = 2*pos + 2;
905     int m = ini + (fim - ini) / 2;
906
907     build(e, ini, m, v);
908     build(d, m + 1, fim, v);
909
910     seg[pos] = f(seg[e], seg[d]);
911 }
912
913 ftype query(int p, int q) {
914     return query(0, 0, n - 1, p, q);
915 }
916
917 void update(int p, int q, int val) {
918     update(0, 0, n - 1, p, q, val);
919 }
920
921 void build(vector<int> &v) {
922     build(0, 0, n - 1, v);
923 }
924
925 void debug() {
926     for (auto e : seg) {
927         cout << e << ' ';
928     }
929     cout << '\n';
930     for (auto e : lazy) {
931         cout << e << ' ';
932     }
933     cout << '\n';
934     cout << '\n';
935 }
936
937 void build(int pos, int ini, int fim, vector<int>
938 &v) {
939     if (ini == fim) {
940         // se a posição existir no array original
941         // seg tamanho potencia de dois
942         if (ini < (int)v.size()) {
943             seg[pos] = v[ini];
944         }
945         return;
946     }
947
948     int e = 2*pos + 1;
949     int d = 2*pos + 2;
950     int m = ini + (fim - ini) / 2;
951
952     build(e, ini, m, v);
953     build(d, m + 1, fim, v);
954
955     seg[pos] = f(seg[e], seg[d]);
956 }
957
958 ftype query(int p, int q) {
959     return query(0, 0, n - 1, p, q);
960 }
961
962 void update(int p, int q, int val) {
963     update(0, 0, n - 1, p, q, val);
964 }
965
966 void build(vector<int> &v) {
967     build(0, 0, n - 1, v);
968 }
969
970 void debug() {
971     for (auto e : seg) {
972         cout << e << ' ';
973     }
974     cout << '\n';
975     for (auto e : lazy) {
976         cout << e << ' ';
977     }
978     cout << '\n';
979     cout << '\n';
980 }
981
982 void build(int pos, int ini, int fim, vector<int>
983 &v) {
984     if (ini == fim) {
985         // se a posição existir no array original
986         // seg tamanho potencia de dois
987         if (ini < (int)v.size()) {
988             seg[pos] = v[ini];
989         }
990         return;
991     }
992
993     int e = 2*pos + 1;
994     int d = 2*pos + 2;
995     int m = ini + (fim - ini) / 2;
996
997     build(e, ini, m, v);
998     build(d, m + 1, fim, v);
999
1000    seg[pos] = f(seg[e], seg[d]);
1001 }
1002
1003 ftype query(int p, int q) {
1004     return query(0, 0, n - 1, p, q);
1005 }
1006
1007 void update(int p, int q, int val) {
1008     update(0, 0, n - 1, p, q, val);
1009 }
1010
1011 void build(vector<int> &v) {
1012     build(0, 0, n - 1, v);
1013 }
1014
1015 void debug() {
1016     for (auto e : seg) {
1017         cout << e << ' ';
1018     }
1019     cout << '\n';
1020     for (auto e : lazy) {
1021         cout << e << ' ';
1022     }
1023     cout << '\n';
1024     cout << '\n';
1025 }
1026
1027 void build(int pos, int ini, int fim, vector<int>
1028 &v) {
1029     if (ini == fim) {
1030         // se a posição existir no array original
1031         // seg tamanho potencia de dois
1032         if (ini < (int)v.size()) {
1033             seg[pos] = v[ini];
1034         }
1035         return;
1036     }
1037
1038     int e = 2*pos + 1;
1039     int d = 2*pos + 2;
1040     int m = ini + (fim - ini) / 2;
1041
1042     build(e, ini, m, v);
1043     build(d, m + 1, fim, v);
1044
1045     seg[pos] = f(seg[e], seg[d]);
1046 }
1047
1048 ftype query(int p, int q) {
1049     return query(0, 0, n - 1, p, q);
1050 }
1051
1052 void update(int p, int q, int val) {
1053     update(0, 0, n - 1, p, q, val);
1054 }
1055
1056 void build(vector<int> &v) {
1057     build(0, 0, n - 1, v);
1058 }
1059
1060 void debug() {
1061     for (auto e : seg) {
1062         cout << e << ' ';
1063     }
1064     cout << '\n';
1065     for (auto e : lazy) {
1066         cout << e << ' ';
1067     }
1068     cout << '\n';
1069     cout << '\n';
1070 }
1071
1072 void build(int pos, int ini, int fim, vector<int>
1073 &v) {
1074     if (ini == fim) {
1075         // se a posição existir no array original
1076         // seg tamanho potencia de dois
1077         if (ini < (int)v.size()) {
1078             seg[pos] = v[ini];
1079         }
1080         return;
1081     }
1082
1083     int e = 2*pos + 1;
1084     int d = 2*pos + 2;
1085     int m = ini + (fim - ini) / 2;
1086
1087     build(e, ini, m, v);
1088     build(d, m + 1, fim, v);
1089
1090     seg[pos] = f(seg[e], seg[d]);
1091 }
1092
1093 ftype query(int p, int q) {
1094     return query(0, 0, n - 1, p, q);
1095 }
1096
1097 void update(int p, int q, int val) {
1098     update(0, 0, n - 1, p, q, val);
1099 }
1100
1101 void build(vector<int> &v) {
1102     build(0, 0, n - 1, v);
1103 }
1104
1105 void debug() {
1106     for (auto e : seg) {
1107         cout << e << ' ';
1108     }
1109     cout << '\n';
1110     for (auto e : lazy) {
1111         cout << e << ' ';
1112     }
1113     cout << '\n';
1114     cout << '\n';
1115 }
1116
1117 void build(int pos, int ini, int fim, vector&lt
```

```

19 // MAX is the maximum index a node can assume
20 // Create a default null node
21 // Create a node to be the root of the segtree
22
23 // Segtree seg = Segtree(MAX);
24 // seg.create();
25 // seg.create();
26
27 const int MAX = 1e9+10;
28 const long long INF = 1e18+10;
29
30 typedef long long ftype;
31
32 struct Segtree {
33     vector<ftype> seg, d, e, lazy;
34     const ftype NEUTRAL = 0;
35     const ftype NEUTRAL_LAZY = -INF;
36     int n;
37
38     Segtree(int n) {
39         this->n = n;
40     }
41
42     ftype apply_lazy(ftype a, ftype b, int len) {
43         if (b == NEUTRAL_LAZY) return a;
44         else return b * len;
45     }
46
47     void propagate(int pos, int ini, int fim) {
48         if (seg[pos] == 0) return;
49
50         if (ini == fim) {
51             return;
52         }
53
54         int m = (ini + fim) >> 1;
55
56         if (e[pos] == 0) e[pos] = create();
57         if (d[pos] == 0) d[pos] = create();
58
59         lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[
pos], 1);
60         lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[
pos], 1);
61
62         seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[
pos], m - ini + 1);
63         seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[
pos], fim - m);
64
65         lazy[pos] = NEUTRAL_LAZY;
66     }
67
68     ftype f(ftype a, ftype b) {
69         return a + b;
70     }
71
72     ftype create() {
73         seg.push_back(0);
74         e.push_back(0);
75         d.push_back(0);
76         lazy.push_back(-1);
77         return seg.size() - 1;
78     }
79
80     ftype query(int pos, int ini, int fim, int p, int
q) {
81         propagate(pos, ini, fim);
82         if (q < ini || p > fim) return NEUTRAL;
83         if (pos == 0) return 0;
84         if (p <= ini && fim <= q) return seg[pos];
85         int m = (ini + fim) >> 1;
86         return f(query(e[pos], ini, m, p, q), query(d

```

```

87         [pos], m + 1, fim, p, q));
88     }
89
90     void update(int pos, int ini, int fim, int p, int
q, int val) {
91         propagate(pos, ini, fim);
92         if (ini > q || fim < p) {
93             return;
94         }
95
96         if (ini >= p && fim <= q) {
97             lazy[pos] = apply_lazy(lazy[pos], val, 1)
;
98             seg[pos] = apply_lazy(seg[pos], val, fim
- ini + 1);
99             return;
100         }
101
102         int m = (ini + fim) >> 1;
103
104         if (e[pos] == 0) e[pos] = create();
105         update(e[pos], ini, m, p, q, val);
106
107         if (d[pos] == 0) d[pos] = create();
108         update(d[pos], m + 1, fim, p, q, val);
109
110         seg[pos] = f(seg[e[pos]], seg[d[pos]]);
111     }
112
113     ftype query(int p, int q) {
114         return query(1, 1, n, p, q);
115     }
116
117     void update(int p, int q, int val) {
118         update(1, 1, n, p, q, val);
119     }
120 };

```

9.13 Persistent

```

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

```

```

22
23 const int MAX = 2e5+17;
24 const int INF = 1e9+17;
25

```



```

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

```