



Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1 Math	2	4.5 Trie	7
1.1 Ceil	2	4.6 Z-function	8
1.2 Matrix Exponentiation	2	5 Misc	8
1.3 Crt	2	5.1 Split	8
1.4 Binary To Decimal	3	5.2 Int128	8
1.5 Fast Exponentiation	3	6 Graphs	8
1.6 Linear Diophantine Equation	3	6.1 Centroid Find	8
1.7 Sieve Of Eratosthenes	4	6.2 Bipartite	9
1.8 Multiplicative Inverse	4	6.3 Prim	9
1.9 Divisors	4	6.4 Ford Fulkerson Edmonds Karp	9
1.10 Check If Bit Is On	4	6.5 Floyd Warshall	10
1.11 Prime Factors	4	6.6 Lca	10
2 DP	4	6.7 Bellman Ford	11
2.1 Knapsack With Index	4	6.8 Dinic	11
2.2 Substr Palindrome	5	6.9 2sat	12
2.3 Edit Distance	5	6.10 Find Cycle	14
2.4 Knapsack	5	6.11 Cycle Path Recovery	14
2.5 Digits	5	6.12 Centroid Decomposition	15
2.6 Coins	6	6.13 Tarjan Bridge	15
2.7 Minimum Coin Change	6	6.14 Small To Large	16
2.8 Kadane	6	6.15 Tree Diameter	16
3 Template	6	6.16 Dijkstra	16
3.1 Template	6	6.17 Kruskall	17
3.2 Template Clean	6	7 Algorithms	17
4 Strings	7	7.1 Lis	17
4.1 Kmp	7	7.2 Delta-encoding	17
4.2 Generate All Permutations	7	7.3 Binary Search Last True	18
4.3 Generate All Sequences Length K	7	7.4 Ternary Search	18
4.4 Lcs	7	7.5 Binary Search First True	18
		7.6 Biggest K	18

8	Data Structures	19
8.1	Ordered Set	19
8.2	Priority Queue	19
8.3	Dsu	19
8.4	Two Sets	20
8.5	Dynamic Implicit Sparse	20
8.6	Segtree2d	21
8.7	Minimum And Amount	22
8.8	Lazy Addition To Segment	23
8.9	Segment With Maximum Sum	24
8.10	Range Query Point Update	25
8.11	Lazy Assignment To Segment	26
8.12	Lazy Dynamic Implicit Sparse	27
8.13	Persistent	28

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 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)
43
44 // How to use:
45 // vector<vector<ll>> v = {{1, 1}, {1, 0}};
46 // Matriz transition = Matriz(v);
47 // cout << fexp(transition, n)[0][1] << '\n';
48
49 using ll = long long;
50
51 const int MOD = 1e9+7;
52
53 struct Matriz{
54     vector<vector<ll>> mat;
55     int rows, columns;
56
57     vector<ll> operator[](int i){
58         return mat[i];
59     }
60
61     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 }
```

1.3 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
8         solution
9         ans = ans + x * ((a - ans) / g) % (b / g) *
10         lcm;
11         lcm = lcm * (b / g);
12         ans = (ans % lcm + lcm) % lcm;
13     }
14     return ans;
15 }
```

```
13 }
```

1.4 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 }
```

1.5 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 }
```

1.6 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);
7 // }
8 // else if (a == 0) {
9 //     if (c % b == 0 && y1 <= c / b && y2 >= c / b)
10 //         ans = (x2 - x1 + 1);
11 //     else ans = 0;
12 // }
13 // else if (b == 0) {
14 //     if (c % a == 0 && x1 <= c / a && x2 >= c / a)
15 //         ans = (y2 - y1 + 1);
16 //     else ans = 0;
17 // }
18 // Careful when a or b are negative or zero
19 // if (ans == -1) ans = find_all_solutions(a, b, c,
20 //     x1, x2, y1, y2);
21 // cout << ans << '\n';
22 // Problems:
```

```
22 // https://www.spoj.com/problems/CEQU/
23 // http://codeforces.com/problemsets/acmsguru/problem
24 // 99999/106
```

```
24
25 // consider trivial case a or b is 0
26 int gcd(int a, int b, int& x, int& y) {
27     if (b == 0) {
28         x = 1;
29         y = 0;
30         return a;
31     }
32     int x1, y1;
33     int d = gcd(b, a % b, x1, y1);
34     x = y1;
35     y = x1 - y1 * (a / b);
36     return d;
37 }
38
39 // x and y are one solution and g is the gcd, all
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);
72     if (x < minx)
73         shift_solution(x, y, a, b, sign_b);
74     if (x > maxx)
75         return 0;
76     int lx1 = x;
77
78     shift_solution(x, y, a, b, (maxx - x) / b);
79     if (x > maxx)
80         shift_solution(x, y, a, b, -sign_b);
81     int rx1 = x;
82
83     shift_solution(x, y, a, b, -(miny - y) / a);
84     if (y < miny)
85         shift_solution(x, y, a, b, -sign_a);
86     if (y > maxy)
87         return 0;
88     int lx2 = x;
89
90     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 }

```

1.7 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 }

```

1.8 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 }

```

1.9 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 }

```

1.10 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 }

```

1.11 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 ão.
3
4 char s[MAX];
5 int calculado[MAX][MAX]; // iniciado com false, ou 0
6 int tabela[MAX][MAX];
7
8 int is_palin(int i, int j){
9     if(calculado[i][j]){
10         return tabela[i][j];
11     }
12     if(i == j) return true;
13     if(i + 1 == j) return s[i] == s[j];
14
15     int ans = false;
16     if(s[i] == s[j]){
17         if(is_palin(i+1, j-1)){
18             ans = true;
19         }
20     }
21     calculado[i][j] = true;
22     tabela[i][j] = ans;
23     return ans;
24 }

```

2.3 Edit Distance

```

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

```

```

29     if (n == 0) return m;
30
31     if (dp[m][n] != -1) return dp[m][n];
32
33     if (str1[m - 1] == str2[n - 1]) return dp[m][n] =
  edit_distance(str1, str2, m - 1, n - 1);
34     return dp[m][n] = 1 + min({edit_distance(str1,
  str2, m, n - 1), edit_distance(str1, str2, m - 1,
  n), edit_distance(str1, str2, m - 1, n - 1)});
35 }

```

2.4 Knapsack

```

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

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

3.2 Template Clean

```
1 // Notes:
2 // Compile and execute
3 // g++ teste.cpp -o teste -std=c++17
4 // ./teste < teste.txt
5
```

```

6 // Print with precision
7 // cout << fixed << setprecision(12) << value << endl
8 ;
9 // File as input and output
10 // freopen("input.txt", "r", stdin);
11 // freopen("output.txt", "w", stdout);
12
13 #include <bits/stdc++.h>
14 using namespace std;
15
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(NULL);
19
20
21     return 0;
22 }
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
2 // em set (de comprimento n) e que tenham tamanho k
3 // sequence = ""
4 vector<string> generate_sequences(char set[], string
5     sequence, int n, int k) {
6     if (k == 0){
7         return { sequence };
8     }
9
10    vector<string> ans;
11    for (int i = 0; i < n; i++) {
12        auto aux = generate_sequences(set, sequence +
13            set[i], n, k - 1);
14        ans.insert(ans.end(), aux.begin(), aux.end());
15    }
16
17    // for (auto e : ans) ans.push_back(e);

```

```

13     }
14
15     return ans;
16 }

```

4.4 Lcs

```

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

```

4.5 Trie

```

1 const int K = 26;
2
3 struct Vertex {
4     int next[K];
5     bool output = false;
6     int p = -1;
7     char pch;
8     int link = -1;
9     int go[K];
10
11     Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
12         fill(begin(next), end(next), -1);
13         fill(begin(go), end(go), -1);
14     }

```



```

15 };
16
17 vector<Vertex> t(1);
18
19 void add_string(string const& s) {
20     int v = 0;
21     for (char ch : s) {
22         int c = ch - 'a';
23         if (t[v].next[c] == -1) {
24             t[v].next[c] = t.size();
25             t.emplace_back(v, ch);
26         }
27         v = t[v].next[c];
28     }
29     t[v].output = true;
30 }
31
32 int go(int v, char ch);
33
34 int get_link(int v) {
35     if (t[v].link == -1) {
36         if (v == 0 || t[v].p == 0)
37             t[v].link = 0;
38         else
39             t[v].link = go(get_link(t[v].p), t[v].pch);
40     }
41     return t[v].link;
42 }
43
44 int go(int v, char ch) {
45     int c = ch - 'a';
46     if (t[v].go[c] == -1) {
47         if (t[v].next[c] != -1)
48             t[v].go[c] = t[v].next[c];
49         else
50             t[v].go[c] = v == 0 ? 0 : go(get_link(v),
51                                         ch);
52     }
53     return t[v].go[c];
54 }

```

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
7 // Problem:
8 // https://cses.fi/problemset/task/2079/
9
10 // Complexity:
11 // O(n)
12
13 // How to use:
14 // get_subtree_size(0);
15 // cout << get_centroid(0) + 1 << endl;
16
17 int n;
18 vector<int> adj[MAX];
19 int subtree_size[MAX];
20
21 int get_subtree_size(int node, int par = -1) {
22     int &res = subtree_size[node];
23     res = 1;
24     for (int i : adj[node]) {
25         if (i == par) continue;
26         res += get_subtree_size(i, node);
27     }
28     return res;
29 }

```

```

28 }
29
30 int get_centroid(int node, int par = -1) {
31     for (int i : adj[node]) {
32         if (i == par) continue;
33
34         if (subtree_size[i] * 2 > n) { return
get_centroid(i, node); }
35     }
36     return node;
37 }
38
39 int main() {
40     cin >> n;
41     for (int i = 0; i < n - 1; i++) {
42         int u, v; cin >> u >> v;
43         u--; v--;
44         adj[u].push_back(v);
45         adj[v].push_back(u);
46     }
47
48     get_subtree_size(0);
49     cout << get_centroid(0) + 1 << endl;
50 }

```

6.2 Bipartite

```

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

```

6.3 Prim

```

1 int n;
2 vector<vector<int>> adj; // adjacency matrix of graph
3 const int INF = 1000000000; // weight INF means there
   is no edge
4
5 struct Edge {

```

```

6     int w = INF, to = -1;
7 };
8
9 void prim() {
10     int total_weight = 0;
11     vector<bool> selected(n, false);
12     vector<Edge> min_e(n);
13     min_e[0].w = 0;
14
15     for (int i=0; i<n; ++i) {
16         int v = -1;
17         for (int j = 0; j < n; ++j) {
18             if (!selected[j] && (v == -1 || min_e[j].
w < min_e[v].w))
19                 v = j;
20         }
21
22         if (min_e[v].w == INF) {
23             cout << "No MST!" << endl;
24             exit(0);
25         }
26
27         selected[v] = true;
28         total_weight += min_e[v].w;
29         if (min_e[v].to != -1)
30             cout << v << " " << min_e[v].to << endl;
31
32         for (int to = 0; to < n; ++to) {
33             if (adj[v][to] < min_e[to].w)
34                 min_e[to] = {adj[v][to], v};
35         }
36     }
37
38     cout << total_weight << endl;
39 }

```

6.4 Ford Fulkerson Edmonds Karp

```

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

```

```

30     }
31 }
32
33 return 0;
34 }
35
36 int maxflow(int s, int t) {
37     int flow = 0;
38     vector<int> parent(n);
39     int new_flow;
40
41     while (new_flow = bfs(s, t, parent)) {
42         flow += new_flow;
43         int cur = t;
44         while (cur != s) {
45             int prev = parent[cur];
46             capacity[prev][cur] -= new_flow;
47             capacity[cur][prev] += new_flow;
48             cur = prev;
49         }
50     }
51
52     return flow;
53 }

```

6.5 Floyd Warshall

```

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

```

6.6 Lca

```

1 // Description:
2 // Find the lowest common ancestor between two nodes
3 // in a tree
4
5 // Problem:
6 // https://cses.fi/problemset/task/1688/
7
8 // Complexity:
9 // O(log n)

```

```

9
10 // How to use:
11 // preprocess(1);
12 // lca(a, b);
13
14 // Notes
15 // To calculate the distance between two nodes use
16 // the following formula
17 // dist[a] + dist[b] - 2*dist[lca(a, b)]
18
19 const int MAX = 2e5+17;
20 const int BITS = 32;
21
22 vector<int> adj[MAX];
23 // vector<pair<int, int>> adj[MAX];
24 // int dist[MAX];
25
26 int timer;
27 vector<int> tin, tout;
28 vector<vector<int>> up;
29
30 void dfs(int v, int p)
31 {
32     tin[v] = ++timer;
33     up[v][0] = p;
34
35     for (int i = 1; i <= BITS; ++i) {
36         up[v][i] = up[up[v][i-1]][i-1];
37     }
38
39     for (auto u : adj[v]) {
40         if (u != p) {
41             dfs(u, v);
42         }
43     }
44
45     /*for (auto [u, peso] : adj[v]) {
46         if (u != p) {
47             dist[u] = dist[v] + peso;
48             dfs(u, v);
49         }
50     }*/
51
52     tout[v] = ++timer;
53 }
54
55 bool is_ancestor(int u, int v)
56 {
57     return tin[u] <= tin[v] && tout[u] >= tout[v];
58 }
59
60 int lca(int u, int v)
61 {
62     if (is_ancestor(u, v))
63         return u;
64     if (is_ancestor(v, u))
65         return v;
66     for (int i = BITS; i >= 0; --i) {
67         if (!is_ancestor(up[u][i], v))
68             u = up[u][i];
69     }
70     return up[u][0];
71 }
72
73 void preprocess(int root) {
74     tin.resize(MAX);
75     tout.resize(MAX);
76     timer = 0;
77     up.assign(MAX, vector<int>(BITS + 1));
78     dfs(root, root);
79 }

```

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     void augment(ll bottleneck) {
70         flow += bottleneck;
71         residual->flow -= bottleneck;
72     }
73
74     void reverse(ll bottleneck) {
75         flow -= bottleneck;
76         residual->flow += bottleneck;
77     }
78
79     bool operator<(const Edge& e) const {
80         return true;
81     }
82 };
83
84 struct Dinic {
85     int source;
86     int sink;
87     int nodes;
88     ll flow;
89     vector<vector<Edge*>> adj;
90     vector<int> level;
91     vector<int> next;
92     vector<int> reach;
93     vector<bool> visited;
94     vector<vector<int>> path;
95
96     Dinic(int source, int sink, int nodes) : source(
97         source), sink(sink), nodes(nodes) {
98         adj.resize(nodes + 1);
99     }
100
101     void add_edge(int from, int to, ll capacity) {
102         Edge* e1 = new Edge(from, to, capacity);
103         Edge* e2 = new Edge(to, from, 0);
104         // Edge* e2 = new Edge(to, from, capacity);
105         e1->residual = e2;
106         e2->residual = e1;
107         adj[from].pb(e1);
108         adj[to].pb(e2);
109     }
110
111     bool bfs() {
112         level.assign(nodes + 1, -1);
113         queue<int> q;
114         q.push(source);
115         level[source] = 0;
116
117         while (!q.empty()) {
118             int node = q.front();
119             q.pop();
120
121             for (auto e : adj[node]) {
122                 if (level[e->to] == -1 && e->
```

```

110     get_capacity() > 0) {
111         level[e->to] = level[e->from] +
112         1;
113         q.push(e->to);
114     }
115 }
116 return level[sink] != -1;
117 }
118
119 ll dfs(int v, ll flow) {
120     if (v == sink)
121         return flow;
122
123     int sz = adj[v].size();
124     for (int i = next[v]; i < sz; i++) {
125         Edge* e = adj[v][i];
126         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     }
137     return 0;
138 }
139
140 ll max_flow() {
141     flow = 0;
142     while(bfs()) {
143         next.assign(nodes + 1, 0);
144         ll sent = -1;
145         while (sent != 0) {
146             sent = dfs(source, INF);
147             flow += sent;
148         }
149     }
150     return flow;
151 }
152
153 void reachable(int v) {
154     visited[v] = true;
155
156     for (auto e : adj[v]) {
157         if (!visited[e->to] && e->get_capacity()
158 > 0) {
159             reach.pb(e->to);
160             visited[e->to] = true;
161             reachable(e->to);
162         }
163     }
164 }
165
166 void print_min_cut() {
167     reach.clear();
168     visited.assign(nodes + 1, false);
169     reach.pb(source);
170     reachable(source);
171
172     for (auto v : reach) {
173         for (auto e : adj[v]) {
174             if (!visited[e->to] && e->
175 get_capacity() == 0) {
176                 cout << e->from << ' ' << e->to
177 << '\n';
178             }
179         }
180     }
181 }
182
183 ll build_path(int v, int id, ll flow) {
184     visited[v] = true;
185     if (v == sink) {
186         return flow;
187     }
188
189     for (auto e : adj[v]) {
190         if (!visited[e->to] && e->get_flow() > 0)
191         {
192             visited[e->to] = true;
193             ll bottleneck = build_path(e->to, id,
194             min(flow, e->get_flow()));
195             if (bottleneck > 0) {
196                 path[id].pb(e->to);
197                 e->reverse(bottleneck);
198                 return bottleneck;
199             }
200         }
201     }
202     return 0;
203 }
204
205 void print_flow_path() {
206     path.clear();
207     ll sent = -1;
208     int id = -1;
209     while (sent != 0) {
210         visited.assign(nodes + 1, false);
211         path.pb(vector<int>{});
212         sent = build_path(source, ++id, INF);
213         path[id].pb(source);
214     }
215     path.pop_back();
216
217     for (int i = 0; i < id; i++) {
218         cout << path[i].size() << '\n';
219         reverse(path[i].begin(), path[i].end());
220         for (auto e : path[i]) {
221             cout << e << ' ';
222         }
223         cout << '\n';
224     }
225 }
226
227 int main() {
228     ios::sync_with_stdio(false);
229     cin.tie(NULL);
230
231     int n, m; cin >> n >> m;
232
233     Dinic dinic = Dinic(1, n, n);
234
235     for (int i = 1; i <= m; i++) {
236         int v, u; cin >> v >> u;
237         dinic.add_edge(v, u, 1);
238     }
239
240     cout << dinic.max_flow() << '\n';
241     // dinic.print_min_cut();
242     // dinic.print_flow_path();
243
244     return 0;
245 }

```

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
74         add_or(get_not(a), get_not(b));
75     }
76
77     void add_xnor(int a, int b) {
78         add_or(get_not(a), b);
79         add_or(a, get_not(b));
80     }
81
82     void departure_time(int v) {
83         visited[v] = true;
84
85         for (auto u : adj[v]) {
86             if (!visited[u]) departure_time(u);
87         }
88
89         departure.pb(mp(++curr, v));
90     }
91
92     void find_component(int v, int component) {
93         scc[v] = component;
94         visited[v] = true;
95
96         for (auto u : rev[v]) {
97             if (!visited[u]) find_component(u,
98 component);
99         }
100     }
101
102     void topological_order(int v) {
103         visited[v] = true;
104
105         for (auto u : condensed[v]) {
106             if (!visited[u]) topological_order(u);
107         }
108
109         order.pb(v);
110     }
111
112     bool is_possible() {
113         component = 0;
114         for (int i = 1; i <= 2 * nodes; i++) {
115             if (!visited[i]) departure_time(i);
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             add_or(get_not(a), get_not(b));
147         }
148     }

```

```

140     for (int i = 1; i <= 2 * nodes; i++) {
141         for (auto u : adj[i]) {
142             if (scc[i] != scc[u]) condensed[scc[i]
]].pb(scc[u]);
143         }
144     }
145
146     visited.assign(component + 1, false);
147
148     for (int i = 1; i <= component; i++) {
149         if (!visited[i]) topological_order(i);
150     }
151
152     reverse(order.begin(), order.end());
153
154     // 0 - false
155     // 1 - true
156     // 2 - no value yet
157     vector<int> ans(2 * nodes + 1, 2);
158
159     vector<vector<int>> belong(component + 1);
160
161     for (int i = 1; i <= 2 * nodes; i++) {
162         belong[scc[i]].pb(i);
163     }
164
165     for (auto p : order) {
166         for (auto e : belong[p]) {
167             ans[e] = find_value(e, ans);
168         }
169     }
170
171     return ans;
172 }
173 };
174
175 int main() {
176     ios::sync_with_stdio(false);
177     cin.tie(NULL);
178
179     int n, m; cin >> n >> m;
180
181     SAT sat = SAT(m);
182
183     for (int i = 0; i < n; i++) {
184         char op1, op2; int a, b; cin >> op1 >> a >>
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
26     visited.reset();
27
28     for (int u = 1; u <= N; ++u){
29         path.clear();
30         if (not visited[u] and dfs(u, -1))
31             return true;
32     }
33
34
35     return false;
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] +
19                    vzs;
20                    sum_num[curr] = item;
21                } else if (colors[curr][item] + vzs ==
22                vmax[curr]) {
23                    sum_num[curr] += item;
24                }
25
26                colors[curr][item] += vzs;
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 }

```

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     }
37 }

```

```

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     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 Kruskal

```

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),
46            weight(weight) {}
47
48        bool operator<(const Edge& other) const {
49            return weight < other.weight;
50        }
51
52        bool operator>(const Edge& other) const {
53            return weight > other.weight;
54        }
55    };
56
57    vector<Edge> kruskal(vector<Edge> edges, int n) {
58        vector<Edge> result; // arestas da MST
59        long long cost = 0;
60
61        sort(edges.begin(), edges.end());
62
63        DSU dsu(n);
64
65        for (auto e : edges) {
66            if (!dsu.same(e.u, e.v)) {
67                cost += e.weight;
68                result.push_back(e);
69                dsu.unite(e.u, e.v);
70            }
71        }
72
73        return result;
74    }

```

7 Algorithms

7.1 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 }

```

7.2 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 }

```

7.3 Binary Search Last True

```

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

```

7.4 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         function at m1
9         function at m2
10        if (f1 < f2)
11            l = m1;
12        else
13            r = m2;
14    }
15    return f(l);                  //return the
16    maximum of f(x) in [l, r]
17 }

```

7.5 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;

```

```

7     } else {
8         lo = mid + 1;
9     }
10 }
11 return lo;
12 }

```

7.6 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/
4 // abc306_e
5 // Complexity: O(log n)
6
7 struct SetSum {
8     ll s = 0;
9     multiset<ll> mt;
10    void add(ll x){
11        mt.insert(x);
12        s += x;
13    }
14    int pop(ll x){
15        auto f = mt.find(x);
16        if(f == mt.end()) return 0;
17        mt.erase(f);
18        s -= x;
19        return 1;
20    }
21 };
22
23 struct BigK {
24     int k;
25     SetSum gt, mt;
26     BigK(int _k){
27         k = _k;
28     }
29     void balancear(){
30         while((int)gt.mt.size() < k && (int)mt.mt.
31         size()){
32             auto p = (prev(mt.mt.end()));
33             gt.add(*p);
34             mt.pop(*p);
35         }
36         while((int)mt.mt.size() && (int)gt.mt.size()
37         &&
38         *(gt.mt.begin()) < *(prev(mt.mt.end())) ){
39             ll u = *(gt.mt.begin());
40             ll v = *(prev(mt.mt.end()));
41             gt.pop(u); mt.pop(v);
42             gt.add(v); mt.add(u);
43         }
44     }
45     void add(ll x){
46         mt.add(x);
47         balancear();
48     }
49     void rem(ll x){
50         //x = -x;
51         if(mt.pop(x) == 0)
52             gt.pop(x);
53         balancear();
54     }
55 };
56
57 int main() {
58     ios::sync_with_stdio(false);
59     cin.tie(NULL);
60
61     int n, k, q; cin >> n >> k >> q;

```

```

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 }

```

8 Data Structures

8.1 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 }

```

8.2 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;

```

8.3 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 }

```

8.4 Two Sets

```

1 // Description
2 // The values are divided in two multisets so that
3 // one of them contain all values that are
4 // smaller than the median and the other one contains
5 // all values that are greater or equal to the
6 // median.
7
8 // Problem:
9 // https://atcoder.jp/contests/abc306/tasks/abc306_e
10 // Problem I - Maratona Feminina de çãProgramao da
11 // Unicamp 2023
12 // https://codeforces.com/group/WYIydkIPyE/contest
13 // /450037/attachments
14
15 // Complexity:
16 // Add and remove elements - O(log n)
17 // Return sum of biggest or smallest set or return
18 // the median - O(1)
19
20 using ll = long long;
21
22 struct TwoSets {
23     multiset<int> small;
24     multiset<int> big;
25     ll sums = 0;
26     ll sumb = 0;
27     int n = 0;
28
29     int size_small() {
30         return small.size();
31     }
32
33     int size_big() {
34         return big.size();
35     }
36
37     void balance() {
38         while (size_small() > n / 2) {
39             int v = *small.rbegin();
40             small.erase(prev(small.end()));
41             big.insert(v);
42             sums -= v;
43             sumb += v;
44         }
45         while (size_big() > n - n / 2) {
46             int v = *big.begin();
47             big.erase(big.begin());
48             small.insert(v);
49             sumb -= v;
50             sums += v;
51         }
52     }
53
54     ll sum_small() {
55         return sums;
56     }
57
58     ll sum_big() {
59         return sumb;
60     }
61
62     int median() {
63         return *big.begin();
64     }
65 };

```

```

43     sumb -= v;
44     sums += v;
45 }
46
47 void add(int x) {
48     n++;
49     small.insert(x);
50     sums += x;
51     while (!small.empty() && *small.rbegin() > *big.
52         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 };

```

8.5 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

```

```

15 // Complexity:
16 // O(log n) for both query and update
17
18 // How to use:
19 // MAX is the maximum index a node can assume
20
21 // Segtree seg = Segtree(MAX);
22
23 typedef long long ftype;
24
25 const int MAX = 1e9+17;
26
27 struct Segtree {
28     vector<ftype> seg, d, e;
29     const ftype NEUTRAL = 0;
30     int n;
31
32     Segtree(int n) {
33         this->n = n;
34         create();
35         create();
36     }
37
38     ftype f(ftype a, ftype b) {
39         return a + b;
40     }
41
42     ftype create() {
43         seg.push_back(0);
44         e.push_back(0);
45         d.push_back(0);
46         return seg.size() - 1;
47     }
48
49     ftype query(int pos, int ini, int fim, int p, int
50     q) {
51         if (q < ini || p > fim) return NEUTRAL;
52         if (pos == 0) return 0;
53         if (p <= ini && fim <= q) return seg[pos];
54         int m = (ini + fim) >> 1;
55         return f(query(e[pos], ini, m, p, q), query(d
56         [pos], m + 1, fim, p, q));
57     }
58
59     void update(int pos, int ini, int fim, int id,
60     int val) {
61         if (ini > id || fim < id) {
62             return;
63         }
64
65         if (ini == fim) {
66             seg[pos] = val;
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 }

```

```

85     void update(int id, int val) {
86         update(1, 1, n, id, val);
87     }
88 };

```

8.6 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 // Segtree2Dseg = 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     vector<vector<int>> seg;
33
34     Segtree2D(int N, int M) {
35         this->N = N;
36         this->M = M;
37         seg.resize(2*MAXN, vector<int>(2*MAXN));
38     }
39
40     void buildY(int noX, int lX, int rX, int noY, int
41     lY, int rY, vector<vector<int>> &v){
42         if(lY == rY){
43             if(lX == rX){
44                 seg[noX][noY] = v[rX][rY];
45             }else{
46                 seg[noX][noY] = seg[2*noX+1][noY] +
47                 seg[2*noX+2][noY];
48             }
49         }else{
50             int m = (lY+rY)/2;
51
52             buildY(noX, lX, rX, 2*noY+1, lY, m, v);
53             buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);
54
55             seg[noX][noY] = seg[noX][2*noY+1] + seg[
56             noX][2*noY+2];
57         }
58     }
59
60     void buildX(int noX, int lX, int rX, vector<
61     vector<int>> &v){
62         if(lX != rX){
63             int m = (lX+rX)/2;
64
65             buildX(2*noX+1, lX, m, v);
66             buildX(2*noX+2, m+1, rX, v);
67         }
68     }
69 }

```

```

61     }
62
63     buildY(noX, lX, rX, 0, 0, M - 1, v);
64 }
65
66 void updateY(int noX, int lX, int rX, int noY,
67 int lY, int rY, int y){
68     if(lY == rY){
69         if(lX == rX){
70             seg[noX][noY] = !seg[noX][noY];
71         }else{
72             seg[noX][noY] = seg[2*noX+1][noY] +
73             seg[2*noX+2][noY];
74         }
75     }else{
76         int m = (lY+rY)/2;
77
78         if(y <= m){
79             updateY(noX, lX, rX, 2*noY+1, lY, m, y
80 );
81         }else if(m < y){
82             updateY(noX, lX, rX, 2*noY+2, m+1, rY
83 , y);
84         }
85
86         seg[noX][noY] = seg[noX][2*noY+1] + seg[
87 noX][2*noY+2];
88     }
89 }
90
91 void updateX(int noX, int lX, int rX, int x, int
92 y){
93     int m = (lX+rX)/2;
94
95     if(lX != rX){
96         if(x <= m){
97             updateX(2*noX+1, lX, m, x, y);
98         }else if(m < x){
99             updateX(2*noX+2, m+1, rX, x, y);
100         }
101     }
102
103     updateY(noX, lX, rX, 0, 0, M - 1, y);
104 }
105
106 int queryY(int noX, int noY, int lY, int rY, int
107 aY, int bY){
108     if(aY <= lY && rY <= bY) return seg[noX][noY
109 ];
110
111     int m = (lY+rY)/2;
112
113     if(bY <= m) return queryY(noX, 2*noY+1, lY, m
114 , aY, bY);
115     if(m < aY) return queryY(noX, 2*noY+2, m+1,
116 rY, aY, bY);
117
118     return queryY(noX, 2*noY+1, lY, m, aY, bY) +
119 queryY(noX, 2*noY+2, m+1, rY, aY, bY);
120 }
121
122 int queryX(int noX, int lX, int rX, int aX, int
123 bX, int aY, int bY){
124     if(aX <= lX && rX <= bX) return queryY(noX,
125 0, 0, M - 1, aY, bY);
126
127     int m = (lX+rX)/2;
128
129     if(bX <= m) return queryX(2*noX+1, lX, m, aX,
130 bX, aY, bY);
131     if(m < aX) return queryX(2*noX+2, m+1, rX, aX
132 , bX, aY, bY);
133 }

```

```

119     return queryX(2*noX+1, lX, m, aX, bX, aY, bY)
120 + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
121 }
122
123 void build(vector<vector<int>> &v) {
124     buildX(0, 0, N - 1, v);
125 }
126
127 int query(int aX, int bX, int aY, int bY) {
128     return queryX(0, 0, N - 1, aX, bX, aY, bY);
129 }
130
131 void update(int x, int y) {
132     updateX(0, 0, N - 1, x, y);
133 }
134 };

```

8.7 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);
19
20 #define pii pair<int, int>
21 #define mp make_pair
22 #define ff first
23 #define ss second
24
25 const int INF = 1e9+17;
26
27 typedef pii ftype;
28
29 struct Segtree {
30     vector<ftype> seg;
31     int n;
32     const ftype NEUTRAL = mp(INF, 0);
33
34     Segtree(int n) {
35         int sz = 1;
36         while (sz < n) sz *= 2;
37         this->n = sz;
38
39         seg.assign(2*sz, NEUTRAL);
40     }
41
42     ftype f(ftype a, ftype b) {
43         if (a.ff < b.ff) return a;
44         if (b.ff < a.ff) return b;
45
46         return mp(a.ff, a.ss + b.ss);
47     }
48
49     ftype query(int pos, int ini, int fim, int p, int
50 q) {
51     if (ini >= p && fim <= q) {
52         return seg[pos];
53     }

```

```

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
62 void update(int pos, int ini, int fim, int id,
63 int val) {
64     if (ini > id || fim < id) {
65         return;
66     }
67
68     if (ini == id && fim == id) {
69         seg[pos] = mp(val, 1);
70
71         return;
72     }
73
74     int e = 2*pos + 1;
75     int d = 2*pos + 2;
76     int m = ini + (fim - ini) / 2;
77
78     update(e, ini, m, id, val);
79     update(d, m + 1, fim, id, val);
80
81     seg[pos] = f(seg[e], seg[d]);
82 }
83
84 void build(int pos, int ini, int fim, vector<int>
85 &v) {
86     if (ini == fim) {
87         if (ini < (int)v.size()) {
88             seg[pos] = mp(v[ini], 1);
89         }
90         return;
91     }
92
93     int e = 2*pos + 1;
94     int d = 2*pos + 2;
95     int m = ini + (fim - ini) / 2;
96
97     build(e, ini, m, v);
98     build(d, m + 1, fim, v);
99
100     seg[pos] = f(seg[e], seg[d]);
101 }
102
103 ftype query(int p, int q) {
104     return query(0, 0, n - 1, p, q);
105 }
106
107 void update(int id, int val) {
108     update(0, 0, n - 1, id, val);
109 }
110
111 void build(vector<int> &v) {
112     build(0, 0, n - 1, v);
113 }
114
115 void debug() {
116     for (auto e : seg) {
117         cout << e.ff << ' ' << e.ss << '\n';
118     }
119     cout << '\n';
120 }
121
122 };

```

8.8 Lazy Addition To Segment

```

1 // Description:
2 // Query - get sum of elements from range (l, r)
   inclusive
3 // Update - add a value val to elementos from range (
   l, r) inclusive
4
5 // Problem:
6 // https://codeforces.com/edu/course/2/lesson/5/1/
   practice/contest/279634/problem/A
7
8 // Complexity:
9 // O(log n) for both query and update
10
11 // How to use:
12 // Segtree seg = Segtree(n);
13 // seg.build(v);
14
15 // Notes
16 // Change neutral element and f function to perform a
   different operation
17
18 const long long INF = 1e18+10;
19
20 typedef long long ftype;
21
22 struct Segtree {
23     vector<ftype> seg;
24     vector<ftype> lazy;
25     int n;
26     const ftype NEUTRAL = 0;
27     const ftype NEUTRAL_LAZY = -INF;
28
29     Segtree(int n) {
30         int sz = 1;
31         while (sz < n) sz *= 2;
32         this->n = sz;
33
34         seg.assign(2*sz, NEUTRAL);
35         lazy.assign(2*sz, NEUTRAL_LAZY);
36     }
37
38     ftype apply_lazy(ftype a, ftype b, int len) {
39         if (b == NEUTRAL_LAZY) return a;
40         if (a == NEUTRAL_LAZY) return b * len;
41         else return a + b * len;
42     }
43
44     void propagate(int pos, int ini, int fim) {
45         if (ini == fim) {
46             return;
47         }
48
49         int e = 2*pos + 1;
50         int d = 2*pos + 2;
51         int m = ini + (fim - ini) / 2;
52
53         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
54         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
55
56         seg[e] = apply_lazy(seg[e], lazy[pos], m -
57 ini + 1);
58         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
59 m);
60
61         lazy[pos] = NEUTRAL_LAZY;
62     }
63
64     ftype f(ftype a, ftype b) {
65         return a + b;
66     }
67 }

```



```

66 ftype query(int pos, int ini, int fim, int p, int q) {
67     propagate(pos, ini, fim);
68
69     if (ini >= p && fim <= q) {
70         return seg[pos];
71     }
72
73     if (q < ini || p > fim) {
74         return NEUTRAL;
75     }
76
77     int e = 2*pos + 1;
78     int d = 2*pos + 2;
79     int m = ini + (fim - ini) / 2;
80
81     return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
82 }
83
84 void update(int pos, int ini, int fim, int p, int q, int val) {
85     propagate(pos, ini, fim);
86
87     if (ini > q || fim < p) {
88         return;
89     }
90
91     if (ini >= p && fim <= q) {
92         lazy[pos] = apply_lazy(lazy[pos], val, 1);
93         seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);
94         return;
95     }
96
97     int e = 2*pos + 1;
98     int d = 2*pos + 2;
99     int m = ini + (fim - ini) / 2;
100
101     update(e, ini, m, p, q, val);
102     update(d, m + 1, fim, p, q, val);
103
104     seg[pos] = f(seg[e], seg[d]);
105 }
106
107 void build(int pos, int ini, int fim, vector<int> &v) {
108     if (ini == fim) {
109         if (ini < (int)v.size()) {
110             seg[pos] = v[ini];
111         }
112         return;
113     }
114
115     int e = 2*pos + 1;
116     int d = 2*pos + 2;
117     int m = ini + (fim - ini) / 2;
118
119     build(e, ini, m, v);
120     build(d, m + 1, fim, v);
121
122     seg[pos] = f(seg[e], seg[d]);
123 }
124
125 ftype query(int p, int q) {
126     return query(0, 0, n - 1, p, q);
127 }
128
129 void update(int p, int q, int val) {
130     update(0, 0, n - 1, p, q, val);
131 }
132

```

```

134 void build(vector<int> &v) {
135     build(0, 0, n - 1, v);
136 }
137
138 void debug() {
139     for (auto e : seg) {
140         cout << e << ' ';
141     }
142     cout << '\n';
143     for (auto e : lazy) {
144         cout << e << ' ';
145     }
146     cout << '\n';
147     cout << '\n';
148 }
149

```

8.9 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
21 // choice
22 // So we need to take the maximum between 0 and the
23 // query
24 // max(0LL, seg.query(0, n).max_seg)
25 using ll = long long;
26 typedef ll ftype_node;
27
28 struct Node {
29     ftype_node max_seg;
30     ftype_node pref;
31     ftype_node suf;
32     ftype_node sum;
33
34     Node(ftype_node max_seg, ftype_node pref,
35          ftype_node suf, ftype_node sum) : max_seg(max_seg),
36          pref(pref), suf(suf), sum(sum) {};
37 };
38
39 typedef Node ftype;
40
41 struct Segtree {
42     vector<ftype> seg;
43     int n;
44     const ftype NEUTRAL = Node(0, 0, 0, 0);
45
46     Segtree(int n) {
47         int sz = 1;
48         // potencia de dois mais proxima

```

```

47     while (sz < n) sz *= 2;
48     this->n = sz;
49
50     // numero de nos da seg
51     seg.assign(2*sz, NEUTRAL);
52 }
53
54 ftype f(ftype a, ftype b) {
55     ftype_node max_seg = max({a.max_seg, b.
max_seg, a.suf + b.pref});
56     ftype_node pref = max(a.pref, a.sum + b.pref);
57
58     ftype_node suf = max(b.suf, b.sum + a.suf);
59     ftype_node sum = a.sum + b.sum;
60
61     return Node(max_seg, pref, suf, sum);
62 }
63
64 ftype query(int pos, int ini, int fim, int p, int
q) {
65     if (ini >= p && fim <= q) {
66         return seg[pos];
67     }
68
69     if (q < ini || p > fim) {
70         return NEUTRAL;
71     }
72
73     int e = 2*pos + 1;
74     int d = 2*pos + 2;
75     int m = ini + (fim - ini) / 2;
76
77     return f(query(e, ini, m, p, q), query(d, m +
1, fim, p, q));
78 }
79
80 void update(int pos, int ini, int fim, int id,
int val) {
81     if (ini > id || fim < id) {
82         return;
83     }
84
85     if (ini == id && fim == id) {
86         seg[pos] = Node(val, val, val, val);
87     }
88
89     return;
90
91     int e = 2*pos + 1;
92     int d = 2*pos + 2;
93     int m = ini + (fim - ini) / 2;
94
95     update(e, ini, m, id, val);
96     update(d, m + 1, fim, id, val);
97
98     seg[pos] = f(seg[e], seg[d]);
99 }
100
101 void build(int pos, int ini, int fim, vector<int>
&v) {
102     if (ini == fim) {
103         // se a çãposio existir no array original
104         // seg tamanho potencia de dois
105         if (ini < (int)v.size()) {
106             seg[pos] = Node(v[ini], v[ini], v[ini
], v[ini]);
107         }
108         return;
109     }
110
111     int e = 2*pos + 1;
112     int d = 2*pos + 2;
113     int m = ini + (fim - ini) / 2;
114
115     build(e, ini, m, v);
116     build(d, m + 1, fim, v);
117
118     seg[pos] = f(seg[e], seg[d]);
119 }
120
121 ftype query(int p, int q) {
122     return query(0, 0, n - 1, p, q);
123 }
124
125 void update(int id, int val) {
126     update(0, 0, n - 1, id, val);
127 }
128
129 void build(vector<int> &v) {
130     build(0, 0, n - 1, v);
131 }
132
133 void debug() {
134     for (auto e : seg) {
135         cout << e.max_seg << ' ' << e.pref << ' '
<< e.suf << ' ' << e.sum << '\n';
136     }
137     cout << '\n';
138 };

```

8.10 Range Query Point Update

```

1 // Description:
2 // Indexed at zero
3 // Query - get sum of elements from range (l, r)
4 // inclusive
5 // Update - update element at position id to a value
6 // val
7 // Problem:
8 // https://codeforces.com/edu/course/2/lesson/4/1/
9 // practice/contest/273169/problem/B
10 // Complexity:
11 // 0(log n) for both query and update
12 // How to use:
13 // Segtree seg = Segtree(n);
14 // seg.build(v);
15 // Notes
16 // Change neutral element and f function to perform a
17 // different operation
18 // If you want to change the operations to point
19 // query and range update
20 // Use the same segtree, but perform the following
21 // operations
22 // Query - seg.query(0, id);
23 // Update - seg.update(l, v); seg.update(r + 1, -v);
24
25 typedef long long ftype;
26
27 struct Segtree {
28     vector<ftype> seg;
29     int n;
30     const ftype NEUTRAL = 0;
31
32     Segtree(int n) {
33         int sz = 1;
34         while (sz < n) sz *= 2;
35         this->n = sz;
36
37         seg.assign(2*sz, NEUTRAL);
38     }

```

```

38 ftype f(ftype a, ftype b) {
39     return a + b;
40 }
41
42 ftype query(int pos, int ini, int fim, int p, int q) {
43     if (ini >= p && fim <= q) {
44         return seg[pos];
45     }
46
47     if (q < ini || p > fim) {
48         return NEUTRAL;
49     }
50
51     int e = 2*pos + 1;
52     int d = 2*pos + 2;
53     int m = ini + (fim - ini) / 2;
54
55     return f(query(e, ini, m, p, q), query(d, m +
56 1, fim, p, q));
57 }
58
59 void update(int pos, int ini, int fim, int id,
60 int val) {
61     if (ini > id || fim < id) {
62         return;
63     }
64
65     if (ini == id && fim == id) {
66         seg[pos] = val;
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>
81 &v) {
82     if (ini == fim) {
83         if (ini < (int)v.size()) {
84             seg[pos] = v[ini];
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     build(e, ini, m, v);
94     build(d, m + 1, fim, v);
95
96     seg[pos] = f(seg[e], seg[d]);
97 }
98
99 ftype query(int p, int q) {
100     return query(0, 0, n - 1, p, q);
101 }
102
103 void update(int id, int val) {
104     update(0, 0, n - 1, id, val);
105 }
106
107 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 };

```

8.11 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
54 q) {
55        propagate(pos, ini, fim);
56
57        if (ini >= p && fim <= q) {
58            return seg[pos];
59        }

```

```

58     if (q < ini || p > fim) {
59         return NEUTRAL;
60     }
61
62     int e = 2*pos + 1;
63     int d = 2*pos + 2;
64     int m = ini + (fim - ini) / 2;
65
66     return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
67 }
68
69 void update(int pos, int ini, int fim, int p, int q, int val) {
70     propagate(pos, ini, fim);
71
72     if (ini > q || fim < p) {
73         return;
74     }
75
76     if (ini >= p && fim <= q) {
77         lazy[pos] = apply_lazy(lazy[pos], val, 1);
78     }
79     else {
80         seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);
81     }
82
83     return;
84
85     int e = 2*pos + 1;
86     int d = 2*pos + 2;
87     int m = ini + (fim - ini) / 2;
88
89     update(e, ini, m, p, q, val);
90     update(d, m + 1, fim, p, q, val);
91
92     seg[pos] = f(seg[e], seg[d]);
93 }
94
95 void build(int pos, int ini, int fim, vector<int> &v) {
96     if (ini == fim) {
97         // se a posição existir no array original
98         // seg tamanho potencia de dois
99         if (ini < (int)v.size()) {
100             seg[pos] = v[ini];
101         }
102         return;
103     }
104
105     int e = 2*pos + 1;
106     int d = 2*pos + 2;
107     int m = ini + (fim - ini) / 2;
108
109     build(e, ini, m, v);
110     build(d, m + 1, fim, v);
111
112     seg[pos] = f(seg[e], seg[d]);
113 }
114
115 ftype query(int p, int q) {
116     return query(0, 0, n - 1, p, q);
117 }
118
119 void update(int p, int q, int val) {
120     update(0, 0, n - 1, p, q, val);
121 }
122
123 void build(vector<int> &v) {
124     build(0, 0, n - 1, v);
125 }
126
127 void debug() {

```

```

126     for (auto e : seg) {
127         cout << e << ' ';
128     }
129     cout << '\n';
130     for (auto e : lazy) {
131         cout << e << ' ';
132     }
133     cout << '\n';
134     cout << '\n';
135 }
136 };

```

8.12 Lazy Dynamic Implicit Sparse

```

1 // Description:
2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
5 // stored in an array
6 // and the queries need to be answered online so we
7 // can't sort the nodes and compress them
8 // we create nodes only when they are needed so there
9 // 'll be (Q*log(MAX)) nodes
10 // where Q is the number of queries and MAX is the
11 // maximum index a node can assume
12
13 // Query - get sum of elements from range (l, r)
14 // inclusive
15 // Update - update element at position id to a value
16 // val
17
18 // Problem:
19 // https://oj.uz/problem/view/IZh012_apple
20
21 // Complexity:
22 // O(log n) for both query and update
23
24 // How to use:
25 // MAX is the maximum index a node can assume
26 // Create a default null node
27 // Create a node to be the root of the segtree
28
29 // Segtree seg = Segtree(MAX);
30 // seg.create();
31 // seg.create();
32
33 const int MAX = 1e9+10;
34 const long long INF = 1e18+10;
35
36 typedef long long ftype;
37
38 struct Segtree {
39     vector<ftype> seg, d, e, lazy;
40     const ftype NEUTRAL = 0;
41     const ftype NEUTRAL_LAZY = -INF;
42     int n;
43
44     Segtree(int n) {
45         this->n = n;
46     }
47
48     ftype apply_lazy(ftype a, ftype b, int len) {
49         if (b == NEUTRAL_LAZY) return a;
50         else return b * len;
51     }
52
53     void propagate(int pos, int ini, int fim) {
54         if (seg[pos] == 0) return;
55
56         if (ini == fim) {
57             return;
58         }
59     }

```

```

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
[pos], m + 1, fim, p, q));
87 }
88
89 void update(int pos, int ini, int fim, int p, int
q, int val) {
90     propagate(pos, ini, fim);
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         seg[pos] = apply_lazy(seg[pos], val, fim
- ini + 1);
98
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 };

```

8.13 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
q) {
49         if (q < ini || p > fim) return NEUTRAL;
50         if (pos == 0) return 0;
51         if (p <= ini && fim <= q) return seg[pos];
52         int m = (ini + fim) >> 1;
53         return f(query(e[pos], ini, m, p, q), query(d
[pos], m + 1, fim, p, q));
54     }
55
56     int update(int pos, int ini, int fim, int id, int
val) {
57         int novo = create();

```

```

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

```