



Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

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

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

1 Math

1.1 Ceil

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

1.2 To Decimal

```
1 const string digits { "0123456789
2     ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
3
4 long long to_decimal(const string& rep, long long
5     base) {
6     long long n = 0;
7
8     for (auto c : rep) {
9         n *= base;
10        n += digits.find(c);
11    }
12
13    return n;
14 }
```

1.3 Matrix Exponentiation

```
1 // Description:
2 // Calculate the nth term of a linear recursion
3
4 // Example Fibonacci:
5 // Given a linear recurrence, for example fibonacci
6 // F(n) = n, x <= 1
7 // F(n) = F(n - 1) + F(n - 2), x > 1
8
9 // The recurrence has two terms, so we can build a
10 // matrix 2 x 1 so that
11 // n + 1 = transition * n
12 // (2 x 1) = (2 x 2) * (2 x 1)
13 // F(n)      = a b * F(n - 1)
14 // F(n - 1)   c d   F(n - 2)
15
16 // Another Example:
17 // Given a grid 3 x n, you want to color it using 3
18 // distinct colors so that
19 // no adjacent place has the same color. In how many
20 // different ways can you do that?
21 // There are 6 ways for the first column to be
22 // colored using 3 distinct colors
23 // ans 6 ways using 2 equal colors and 1 distinct one
24
25 // Adding another column, there are:
26 // 3 ways to go from 2 equal to 2 equal
27 // 2 ways to go from 2 equal to 3 distinct
28 // 2 ways to go from 3 distinct to 2 equal
29 // 2 ways to go from 3 distinct to 3 distinct
30
31 // So we star with matrix 6 6 and multiply it by the
32 // transition 3 2 and get 18 12
33 //
34 //      2 2      12 12
35 // the we can exponentiate this matrix to find the
36 // nth column
37
38 // Problem:
39 // https://cses.fi/problemset/task/1722/
40
41 // Complexity:
42 // O(log n)
```

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

```
105 }
```

1.4 Crt

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

1.5 Binary To Decimal

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

1.6 Fast Exponentiation

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

1.7 Linear Diophantine Equation

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

```
6 // }
7 // else if (a == 0) {
8 //     if (c % b == 0 && y1 <= c / b && y2 >= c / b)
9 //         ans = (x2 - x1 + 1);
10 //     else ans = 0;
11 // }
12 // else if (b == 0) {
13 //     if (c % a == 0 && x1 <= c / a && x2 >= c / a)
14 //         ans = (y2 - y1 + 1);
15 //     else ans = 0;
16 // }
17 // Careful when a or b are negative or zero
18 // if (ans == -1) ans = find_all_solutions(a, b, c,
19 //     x1, x2, y1, y2);
20 // cout << ans << '\n';
21 // Problems:
22 // https://www.spoj.com/problems/CEQU/
23 // http://codeforces.com/problemsets/acmsguru/problem
24 // 99999/106
25 // consider trivial case a or b is 0
26 int gcd(int a, int b, int& x, int& y) {
27     if (b == 0) {
28         x = 1;
29         y = 0;
30         return a;
31     }
32     int x1, y1;
33     int d = gcd(b, a % b, x1, y1);
34     x = y1;
35     y = x1 - y1 * (a / b);
36     return d;
37 }
38
39 // x and y are one solution and g is the gcd, all
40 // passed as reference
41 // minx <= x <= maxx miny <= y <= maxy
42 bool find_any_solution(int a, int b, int c, int &x0,
43     int &y0, int &g) {
44     g = gcd(abs(a), abs(b), x0, y0);
45     if (c % g) {
46         return false;
47     }
48     x0 *= c / g;
49     y0 *= c / g;
50     if (a < 0) x0 = -x0;
51     if (b < 0) y0 = -y0;
52     return true;
53 }
54 void shift_solution(int &x, int &y, int a, int b,
55     int cnt) {
56     x += cnt * b;
57     y -= cnt * a;
58 }
59 // return number of solutions in the interval
60 int find_all_solutions(int a, int b, int c, int minx,
61     int maxx, int miny, int maxy) {
62     int x, y, g;
63     if (!find_any_solution(a, b, c, x, y, g))
64         return 0;
65     a /= g;
66     b /= g;
67
68     int sign_a = a > 0 ? +1 : -1;
69     int sign_b = b > 0 ? +1 : -1;
70
71     shift_solution(x, y, a, b, (minx - x) / b);
```

```

71     if (x < minx)
72         shift_solution(x, y, a, b, sign_b);
73     if (x > maxx)
74         return 0;
75     int lx1 = x;
76
77     shift_solution(x, y, a, b, (maxx - x) / b);
78     if (x > maxx)
79         shift_solution(x, y, a, b, -sign_b);
80     int rx1 = x;
81
82     shift_solution(x, y, a, b, -(miny - y) / a);
83     if (y < miny)
84         shift_solution(x, y, a, b, -sign_a);
85     if (y > maxy)
86         return 0;
87     int lx2 = x;
88
89     shift_solution(x, y, a, b, -(maxy - y) / a);
90     if (y > maxy)
91         shift_solution(x, y, a, b, sign_a);
92     int rx2 = x;
93
94     if (lx2 > rx2)
95         swap(lx2, rx2);
96     int lx = max(lx1, lx2);
97     int rx = min(rx1, rx2);
98
99     if (lx > rx)
100         return 0;
101     return (rx - lx) / abs(b) + 1;
102 }

```

1.8 Function Root

```

1  const ld EPS1 = 1e-9; // iteration precision error
2  const ld EPS2 = 1e-4; // output precision error
3
4  ld f(ld x) {
5      // exp(-x) == e^(-x)
6      return p * exp(-x) + q * sin(x) + r * cos(x) + s *
7          tan(x) + t * x * x + u;
8  }
9
10 ld root(ld a, ld b) {
11     while (b - a >= EPS1) {
12         ld c = (a + b) / 2.0;
13         ld y = f(c);
14
15         if (y < 0) b = c;
16         else a = c;
17     }
18     return (a + b) / 2;
19 }
20
21 int main() {
22     ld ans = root(0, 1);
23     if (abs(f(ans)) <= EPS2) cout << fixed <<
24         setprecision(4) << ans << '\n';
25     else cout << "No solution\n";
26
27     return 0;
28 }

```

1.9 Sieve Of Eratosthenes

```

1  vector<bool> is_prime(MAX, true);
2  vector<int> primes;
3
4  void sieve() {
5      is_prime[0] = is_prime[1] = false;

```

```

6      for (int i = 2; i < MAX; i++) {
7          if (is_prime[i]) {
8              primes.push_back(i);
9
10             for (int j = i + i; j < MAX; j += i)
11                 is_prime[j] = false;
12         }
13     }
14 }

```

1.10 Horner Algorithm

```

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

```

1.11 Multiplicative Inverse

```

1  ll extend_euclid(ll a, ll b, ll &x, ll &y) {
2      if (a == 0)
3          {
4              x = 0; y = 1;
5              return b;
6          }
7      ll x1, y1;
8      ll d = extend_euclid(b%a, a, x1, y1);
9      x = y1 - (b / a) * x1;
10     y = x1;
11     return d;
12 }
13
14 // gcd(a, m) = 1 para existir solucao
15 // ax + my = 1, ou a*x = 1 (mod m)
16 ll inv_gcd(ll a, ll m) { // com gcd
17     ll x, y;
18     extend_euclid(a, m, x, y);
19     return ((x % m) + m) % m;
20 }
21
22 ll inv(ll a, ll phim) { // com phi(m), se m for primo
23     // entao phi(m) = p-1
24     ll e = phim-1;
25     return fexp(a, e, MOD);
26 }

```

1.12 Representation Arbitrary Base

```

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

```

1.13 Set Operations

```

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

```

1.14 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 }

```

1.15 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 }

```

1.16 Prime Factors

```

1 vector<pair<long long, int>> fatora(long long n) {
2     vector<pair<long long, int>> ans;
3     for(long long p = 2; p*p <= n; p++) {
4         if(n % p == 0) {
5             int expoente = 0;
6             while(n % p == 0) {

```

```

7                 n /= p;
8                 expoente++;
9             }
10            ans.emplace_back(p, expoente);
11        }
12    }
13    if(n > 1) ans.emplace_back(n, 1);
14    return ans;
15 }

```

2 DP

2.1 Knapsack With Index

```

1 void knapsack(int W, int wt[], int val[], int n) {
2     int i, w;
3     int K[n + 1][W + 1];
4
5     for (i = 0; i <= n; i++) {
6         for (w = 0; w <= W; w++) {
7             if (i == 0 || w == 0)
8                 K[i][w] = 0;
9             else if (wt[i - 1] <= w)
10                K[i][w] = max(val[i - 1] +
11                    K[i - 1][w - wt[i - 1]], K[i -
12                1][w]);
13            else
14                K[i][w] = K[i - 1][w];
15        }
16    }
17
18    int res = K[n][W];
19    cout<< res << endl;
20
21    w = W;
22    for (i = n; i > 0 && res > 0; i--) {
23        if (res == K[i - 1][w])
24            continue;
25        else {
26            cout<<" "<<wt[i - 1] ;
27            res = res - val[i - 1];
28            w = w - wt[i - 1];
29        }
30    }
31
32    int main()
33    {
34        int val[] = { 60, 100, 120 };
35        int wt[] = { 10, 20, 30 };
36        int W = 50;
37        int n = sizeof(val) / sizeof(val[0]);
38
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 índices i e j
2 // é um palindromo ou ãno.
3
4 char s[MAX];
5 int calculado[MAX][MAX]; // inciado com false, ou 0
6 int tabela[MAX][MAX];
7
8 int is_palin(int i, int j){
9     if(calculado[i][j]){
10        return tabela[i][j];

```

```

11 }
12 if(i == j) return true;
13 if(i + 1 == j) return s[i] == s[j];
14
15 int ans = false;
16 if(s[i] == s[j]){
17     if(is_palin(i+1, j-1)){
18         ans = true;
19     }
20 }
21 calculado[i][j] = true;
22 tabela[i][j] = ans;
23 return ans;
24 }

```

2.3 Edit Distance

```

1 // Description:
2 // Minimum number of operations required to transform
  a string into another
3 // Operations allowed: add character, remove
  character, replace character
4
5 // Parameters:
6 // str1 - string to be transformed into str2
7 // str2 - string that str1 will be transformed into
8 // m - size of str1
9 // n - size of str2
10
11 // Problem:
12 // https://cses.fi/problemset/task/1639
13
14 // Complexity:
15 // O(m x n)
16
17 // How to use:
18 // memset(dp, -1, sizeof(dp));
19 // string a, b;
20 // edit_distance(a, b, (int)a.size(), (int)b.size());
21
22 // Notes:
23 // Size of dp matriz is m x n
24
25 int dp[MAX][MAX];
26
27 int edit_distance(string &str1, string &str2, int m,
  int n) {
28     if (m == 0) return n;
29     if (n == 0) return m;
30
31     if (dp[m][n] != -1) return dp[m][n];
32
33     if (str1[m - 1] == str2[n - 1]) return dp[m][n] =
      edit_distance(str1, str2, m - 1, n - 1);
34     return dp[m][n] = 1 + min({edit_distance(str1,
      str2, m, n - 1), edit_distance(str1, str2, m - 1,
      n), edit_distance(str1, str2, m - 1, n - 1)});
35 }

```

2.4 Knapsack

```

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

```

```

11                dp[i][j] = dp[i-1][j];
12            }
13        }
14        return dp[n][m];
15 }

```

2.5 Digits

```

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

```

2.6 Coins

```

1 int tb[1005];
2 int n;
3 vector<int> moedas;
4
5 int dp(int i){
6     if(i >= n)
7         return 0;
8     if(tb[i] != -1)
9         return tb[i];
10
11     tb[i] = max(dp(i+1), dp(i+2) + moedas[i]);
12     return tb[i];
13 }
14
15 int main(){
16     memset(tb, -1, sizeof(tb));
17 }

```

2.7 Minimum Coin Change

```

1 int n;
2 vector<int> valores;

```

```

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

```

2.8 Kadane

```

1 // achar uma subsequencia continua no array que a
  soma seja a maior possivel
2 // nesse caso vc precisa multiplicar exatamente 1
  elemento da subsequencia
3 // e achar a maior soma com isso
4
5 int n, x, arr[MAX], tab[MAX][2]; // tab[maior
  resposta no intervalo][foi multiplicado ou ão]
6
7 int dp(int i, bool mult) {
8     if (i == n-1) {
9         if (!mult) return arr[n-1]*x;
10        return arr[n-1];
11    }
12    if (tab[i][mult] != -1) return tab[i][mult];
13
14    int res;
15
16    if (mult) {
17        res = max(arr[i], arr[i] + dp(i+1, 1));
18    }
19    else {
20        res = max({
21            arr[i]*x,
22            arr[i]*x + dp(i+1, 1),
23            arr[i] + dp(i+1, 0)
24        });
25    }
26
27    return tab[i][mult] = res;
28 }
29
30 int main() {
31
32    memset(tab, -1, sizeof(tab));
33
34    int ans = -oo;
35    for (int i = 0; i < n; i++) {
36        ans = max(ans, dp(i, 0));
37    }
38
39    return 0;
40 }
41
42
43
44 int ans = a[0], ans_l = 0, ans_r = 0;
45 int sum = 0, minus_pos = -1;
46
47 for (int r = 0; r < n; ++r) {
48     sum += a[r];
49     if (sum > ans) {
50         ans = sum;
51         ans_l = minus_pos + 1;

```

```

52         ans_r = r;
53     }
54     if (sum < 0) {
55         sum = 0;
56         minus_pos = r;
57     }
58 }

```

3 Template

3.1 Template

```

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

```

3.2 Template Clean

```

1 // Notes:
2 // Compile and execute
3 // g++ teste.cpp -o teste -std=c++17
4 // ./teste < teste.txt
5
6 // Print with precision
7 // cout << fixed << setprecision(12) << value << endl
  ;
8
9 // File as input and output
10 // freopen("input.txt", "r", stdin);
11 // freopen("output.txt", "w", stdout);
12
13 #include <bits/stdc++.h>
14 using namespace std;
15
16 int main() {
17     ios::sync_with_stdio(false);
18     cin.tie(NULL);
19
20
21
22     return 0;
23 }

```

4 Strings

4.1 Kmp


```

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

```

4.2 Generate All Permutations

```

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

```

4.3 Generate All Sequences Length K

```

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

```

4.4 Lcs

```

1 // Description:
2 // Finds the longest common subsequence between two
   string
3
4 // Problem:
5 // https://codeforces.com/gym/103134/problem/B
6
7 // Complexity:
8 // O(mn) where m and n are the length of the strings
9
10 string lcsAlgo(string s1, string s2, int m, int n) {
11     int LCS_table[m + 1][n + 1];
12
13     for (int i = 0; i <= m; i++) {
14         for (int j = 0; j <= n; j++) {
15             if (i == 0 || j == 0)
16                 LCS_table[i][j] = 0;
17             else if (s1[i - 1] == s2[j - 1])

```

```

18                 LCS_table[i][j] = LCS_table[i - 1][j - 1] +
19                 1;
20             else
21                 LCS_table[i][j] = max(LCS_table[i - 1][j],
22                 LCS_table[i][j - 1]);
23         }
24     }
25
26     int index = LCS_table[m][n];
27     char lcsAlgo[index + 1];
28     lcsAlgo[index] = '\0';
29
30     int i = m, j = n;
31     while (i > 0 && j > 0) {
32         if (s1[i - 1] == s2[j - 1]) {
33             lcsAlgo[index - 1] = s1[i - 1];
34             i--;
35             j--;
36             index--;
37         }
38         else if (LCS_table[i - 1][j] > LCS_table[i][j -
39         1])
40             i--;
41         else
42             j--;
43     }
44     return lcsAlgo;
45 }

```

4.5 Trie

```

1 const int K = 26;
2
3 struct Vertex {
4     int next[K];
5     bool output = false;
6     int p = -1;
7     char pch;
8     int link = -1;
9     int go[K];
10
11     Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
12         fill(begin(next), end(next), -1);
13         fill(begin(go), end(go), -1);
14     }
15 };
16
17 vector<Vertex> t(1);
18
19 void add_string(string const& s) {
20     int v = 0;
21     for (char ch : s) {
22         int c = ch - 'a';
23         if (t[v].next[c] == -1) {
24             t[v].next[c] = t.size();
25             t.emplace_back(v, ch);
26         }
27         v = t[v].next[c];
28     }
29     t[v].output = true;
30 }
31
32 int go(int v, char ch);
33
34 int get_link(int v) {
35     if (t[v].link == -1) {
36         if (v == 0 || t[v].p == 0)
37             t[v].link = 0;
38         else
39             t[v].link = go(get_link(t[v].p), t[v].pch
40             );
41     }
42 }

```

```

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 // Problem:
7 // https://cses.fi/problemset/task/2079/
8 // Complexity:
9 // O(n)
10 // How to use:
11 // get_subtree_size(0);
12 // cout << get_centroid(0) + 1 << endl;
13
14 int n;
15 vector<int> adj[MAX];
16 int subtree_size[MAX];
17
18 int get_subtree_size(int node, int par = -1) {
19     int &res = subtree_size[node];
20     res = 1;
21     for (int i : adj[node]) {
22         if (i == par) continue;
23         res += get_subtree_size(i, node);
24     }
25     return res;
26 }
27
28 int get_centroid(int node, int par = -1) {
29     for (int i : adj[node]) {
30         if (i == par) continue;
31         if (subtree_size[i] * 2 > n) { return
32             get_centroid(i, node); }
33     }
34     return node;
35 }
36
37 int main() {
38     cin >> n;
39     for (int i = 0; i < n - 1; i++) {
40         int u, v; cin >> u >> v;
41         u--; v--;
42         adj[u].push_back(v);
43         adj[v].push_back(u);
44     }
45
46     get_subtree_size(0);
47     cout << get_centroid(0) + 1 << endl;
48 }

```

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
36     return true;
37 }

```

6.3 Prim

```

1  int n;
2  vector<vector<int>> adj; // adjacency matrix of graph
3  const int INF = 1000000000; // weight INF means there
   is no edge
4
5  struct Edge {
6      int w = INF, to = -1;
7  };
8
9  void prim() {
10     int total_weight = 0;
11     vector<bool> selected(n, false);
12     vector<Edge> min_e(n);
13     min_e[0].w = 0;
14
15     for (int i=0; i<n; ++i) {
16         int v = -1;
17         for (int j = 0; j < n; ++j) {
18             if (!selected[j] && (v == -1 || min_e[j].
19 w < min_e[v].w))
20                 v = j;
21         }
22
23         if (min_e[v].w == INF) {
24             cout << "No MST!" << endl;
25             exit(0);
26         }
27
28         selected[v] = true;
29         total_weight += min_e[v].w;
30         if (min_e[v].to != -1)
31             cout << v << " " << min_e[v].to << endl;
32
33         for (int to = 0; to < n; ++to) {

```

```

33             if (adj[v][to] < min_e[to].w)
34                 min_e[to] = {adj[v][to], v};
35         }
36     }
37
38     cout << total_weight << endl;
39 }

```

6.4 Ford Fulkerson Edmonds Karp

```

1  // Description:
2  // Obtains the maximum possible flow rate given a
   network. A network is a graph with a single
   source vertex and a single sink vertex in which
   each edge has a capacity
3
4  // Complexity:
5  //  $O(V * E^2)$  where V is the number of vertex and E
   is the number of edges
6
7  int n;
8  vector<vector<int>> capacity;
9  vector<vector<int>> adj;
10
11 int bfs(int s, int t, vector<int>& parent) {
12     fill(parent.begin(), parent.end(), -1);
13     parent[s] = -2;
14     queue<pair<int, int>> q;
15     q.push({s, INF});
16
17     while (!q.empty()) {
18         int cur = q.front().first;
19         int flow = q.front().second;
20         q.pop();
21
22         for (int next : adj[cur]) {
23             if (parent[next] == -1 && capacity[cur][
24 next]) {
25                 parent[next] = cur;
26                 int new_flow = min(flow, capacity[cur
27 ][next]);
28                 if (next == t)
29                     return new_flow;
30                 q.push({next, new_flow});
31             }
32         }
33     }
34
35     return 0;
36 }
37
38 int maxflow(int s, int t) {
39     int flow = 0;
40     vector<int> parent(n);
41     int new_flow;
42
43     while (new_flow = bfs(s, t, parent)) {
44         flow += new_flow;
45         int cur = t;
46         while (cur != s) {
47             int prev = parent[cur];
48             capacity[prev][cur] -= new_flow;
49             capacity[cur][prev] += new_flow;
50             cur = prev;
51         }
52     }
53
54     return flow;
55 }

```

6.5 Floyd Warshall

```

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

```

6.6 Lca

```

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

```

```

38         auto [v, depth] = q.front();
39         q.pop();
40         level[v] = depth;
41
42         for (auto [u, d] : adj[v]) {
43             if (!visited[u]) {
44                 visited[u] = true;
45                 up[u][0] = v;
46                 q.push(mp(u, depth + 1));
47             }
48         }
49     }
50 }
51
52 void find_level_peso() {
53     queue<pii> q;
54
55     q.push(mp(1, 0));
56     visited[1] = true;
57
58     while (!q.empty()) {
59         auto [v, depth] = q.front();
60         q.pop();
61         level_peso[v] = depth;
62
63         for (auto [u, d] : adj[v]) {
64             if (!visited[u]) {
65                 visited[u] = true;
66                 up[u][0] = v;
67                 q.push(mp(u, depth + d));
68             }
69         }
70     }
71 }
72
73 int lca(int a, int b) {
74     // get the nodes to the same level
75     int mn = min(level[a], level[b]);
76
77     for (int j = 0; j <= BITS; j++) {
78         if (a != -1 && ((level[a] - mn) & (1 << j))) a
79             = up[a][j];
80         if (b != -1 && ((level[b] - mn) & (1 << j))) b
81             = up[b][j];
82     }
83
84     // special case
85     if (a == b) return a;
86
87     // binary search
88     for (int j = BITS; j >= 0; j--) {
89         if (up[a][j] != up[b][j]) {
90             a = up[a][j];
91             b = up[b][j];
92         }
93     }
94     return up[a][0];
95 }
96
97 void preprocess() {
98     visited = vector<bool>(MAX, false);
99     find_level();
100     visited = vector<bool>(MAX, false);
101     find_level_peso();
102
103     for (int j = 1; j <= BITS; j++) {
104         for (int i = 1; i <= n; i++) {
105             if (up[i][j - 1] != -1) up[i][j] = up[up[i][j -
106 1][j - 1];
107         }
108     }
109 }

```

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     }

```

```

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

7.1 2d

```

1 #define vp vector<point>
2 #define ld long double
3 const ld EPS = 1e-6;
4 const ld PI = acos(-1);
5
6 // typedef ll cod;
7 // bool eq(cod a, cod b){ return (a==b); }
8 typedef ld cod;
9 bool eq(cod a, cod b){ return abs(a - b) <= EPS; }
10
11 struct point{
12     cod x, y;
13     int id;
14     point(cod x=0, cod y=0): x(x), y(y){}
15
16     point operator+(const point &o) const{ return {x+
17         o.x, y+o.y}; }
18     point operator-(const point &o) const{ return {x-
19         o.x, y-o.y}; }
20     point operator*(cod t) const{ return {x*t, y*t}; }
21     point operator/(cod t) const{ return {x/t, y/t}; }
22 }

```

```

20     cod operator*(const point &o) const{ return x * o.x + y * o.y; }
21     cod operator^(const point &o) const{ return x * o.y - y * o.x; }
22     bool operator<(const point &o) const{
23         return (eq(x, o.x) ? y < o.y : x < o.x);
24     }
25     bool operator==(const point &o) const{
26         return eq(x, o.x) and eq(y, o.y);
27     }
28     friend ostream& operator<<(ostream& os, point p) {
29         return os << "(" << p.x << ", " << p.y << ")"; }
30 };
31
32 int ccw(point a, point b, point e){ // -1=dir; 0=
33     collinear; 1=esq;
34     cod tmp = (b-a) ^ (e-a); // vector from a to b
35     return (tmp > EPS) - (tmp < -EPS);
36 }
37 ld norm(point a){ // Modulo
38     return sqrt(a * a);
39 }
40 cod norm2(point a){
41     return a * a;
42 }
43 bool nulo(point a){
44     return (eq(a.x, 0) and eq(a.y, 0));
45 }
46 point rotccw(point p, ld a){
47     // a = PI*a/180; // graus
48     return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+p.x*sin(a)));
49 }
50 point rot90cw(point a) { return point(a.y, -a.x); };
51 point rot90ccw(point a) { return point(-a.y, a.x); };
52
53 ld proj(point a, point b){ // a sobre b
54     return a*b/norm(b);
55 }
56 ld angle(point a, point b){ // em radianos
57     ld ang = a*b / norm(a) / norm(b);
58     return acos(max(min(ang, (ld)1), (ld)-1));
59 }
60 ld angle_vec(point v){
61     // return 180/PI*atan2(v.x, v.y); // graus
62     return atan2(v.x, v.y);
63 }
64 ld order_angle(point a, point b){ // from a to b ccw
65     (a in front of b)
66     ld aux = angle(a,b)*180/PI;
67     return ((a^b)<=0 ? aux:360-aux);
68 }
69 bool angle_less(point a1, point b1, point a2, point
70     b2){ // ang(a1,b1) <= ang(a2,b2)
71     point p1((a1*b1), abs((a1^b1)));
72     point p2((a2*b2), abs((a2^b2)));
73     return (p1^p2) <= 0;
74 }
75 ld area(vp &p){ // (points sorted)
76     ld ret = 0;
77     for(int i=2;i<(int)p.size();i++)
78         ret += (p[i]-p[0])^(p[i-1]-p[0]);
79     return abs(ret/2);
80 }
81 ld areaT(point &a, point &b, point &c){
82     return abs((b-a)^(c-a))/2.0;
83 }
84 point center(vp &A){
85     point c = point();
86     int len = A.size();
87     for(int i=0;i<len;i++)
88         c=c+A[i];
89     return c/len;
90 }
91
92 point forca_mod(point p, ld m){
93     ld cm = norm(p);
94     if(cm<EPS) return point();
95     return point(p.x*m/cm,p.y*m/cm);
96 }
97
98 ld param(point a, point b, point v){
99     // v = t*(b-a) + a // return t;
100    // assert(line(a, b).inside_seg(v));
101    return ((v-a) * (b-a)) / ((b-a) * (b-a));
102 }
103
104 bool simetric(vp &a){ //ordered
105     int n = a.size();
106     point c = center(a);
107     if(n&1) return false;
108     for(int i=0;i<n/2;i++)
109         if(ccw(a[i], a[i+n/2], c) != 0)
110             return false;
111     return true;
112 }
113
114 point mirror(point m1, point m2, point p){
115     // mirror point p around segment m1m2
116     point seg = m2-m1;
117     ld t0 = ((p-m1)*seg) / (seg*seg);
118     point ort = m1 + seg*t0;
119     point pm = ort-(p-ort);
120     return pm;
121 }
122
123 ///////////////
124 // Line //
125 ///////////////
126
127 struct line{
128     point p1, p2;
129     cod a, b, c; // ax+by+c = 0;
130     // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
131     line(point p1=0, point p2=0): p1(p1), p2(p2){
132         a = p1.y - p2.y;
133         b = p2.x - p1.x;
134         c = p1 ^ p2;
135     }
136
137     line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
138     {
139         // Gera os pontos p1 p2 dados os coeficientes
140         // isso aqui eh um lixo mas quebra um galho
141         kkkkkk
142         if(b==0){
143             p1 = point(1, -c/a);
144             p2 = point(0, -c/a);
145         }else{
146             p1 = point(1, (-c-a*1)/b);
147             p2 = point(0, -c/b);
148         }
149     }
150
151     cod eval(point p){
152         return a*p.x+b*p.y+c;
153     }
154
155     bool inside(point p){
156         return eq(eval(p), 0);
157     }
158
159     point normal(){
160         return point(a, b);
161     }
162 }

```

```

158     bool inside_seg(point p){
159         return (
160             ((p1-p) ^ (p2-p)) == 0 and
161             ((p1-p) * (p2-p)) <= 0
162         );
163     };
164 }
165 };
166 };
167 // be careful with precision error
168 vp inter_line(line l1, line l2){
169     ld det = l1.a*l2.b - l1.b*l2.a;
170     if(det==0) return {};
171     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
172     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
173     return {point(x, y)};
174 }
175 };
176 // segments not collinear
177 vp inter_seg(line l1, line l2){
178     vp ans = inter_line(l1, l2);
179     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.
180     inside_seg(ans[0]))
181         return {};
182     return ans;
183 }
184 bool seg_has_inter(line l1, line l2){
185     return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.
186     p2, l2.p2) < 0 and
187     ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.
188     p2, l1.p2) < 0;
189 }
190 ld dist_seg(point p, point a, point b){ // point -
191     seg
192     if((p-a)*(b-a) < EPS) return norm(p-a);
193     if((p-b)*(a-b) < EPS) return norm(p-b);
194     return abs((p-a)^(b-a)) / norm(b-a);
195 }
196 ld dist_line(point p, line l){ // point - line
197     return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
198 }
199 line bisector(point a, point b){
200     point d = (b-a)*2;
201     return line(d.x, d.y, a*a - b*b);
202 }
203 }
204 line perpendicular(line l, point p){ // passes
205     through p
206     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
207 }
208 }
209 // Circle //
210 // Circle //
211 // Circle //
212 struct circle{
213     point c; cod r;
214     circle() : c(0, 0), r(0){}
215     circle(const point o) : c(o), r(0){}
216     circle(const point a, const point b){
217         c = (a+b)/2;
218         r = norm(a-c);
219     }
220     circle(const point a, const point b, const point
221     cc){
222         assert(ccw(a, b, cc) != 0);
223         c = inter_line(bisector(a, b), bisector(b, c
224         ))[0];
225         r = norm(a-c);
226     }
227     bool inside(const point &a) const{
228         return norm(a - c) <= r + EPS;
229     }
230 };
231 pair<point, point> tangent_points(circle cr, point p)
232 {
233     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
234     point p1 = rotccw(cr.c-p, -theta);
235     point p2 = rotccw(cr.c-p, theta);
236     assert(d1 >= cr.r);
237     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
238     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
239     return {p1, p2};
240 }
241 circle incircle(point p1, point p2, point p3){
242     ld m1 = norm(p2-p3);
243     ld m2 = norm(p1-p3);
244     ld m3 = norm(p1-p2);
245     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
246     ld s = 0.5*(m1+m2+m3);
247     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
248     return circle(c, r);
249 }
250 circle circumcircle(point a, point b, point c) {
251     circle ans;
252     point u = point((b-a).y, -(b-a).x);
253     point v = point((c-a).y, -(c-a).x);
254     point n = (c-b)*0.5;
255     ld t = (u^v)/(v^u);
256     ans.c = ((a+c)*0.5) + (v*t);
257     ans.r = norm(ans.c-a);
258     return ans;
259 }
260 vp inter_circle_line(circle C, line L){
261     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.
262     p1)*(ab) / (ab*ab));
263     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
264     / (ab*ab);
265     if (h2 < -EPS) return {};
266     if (eq(h2, 0)) return {p};
267     point h = (ab/norm(ab)) * sqrt(h2);
268     return {p - h, p + h};
269 }
270 vp inter_circle(circle C1, circle C2){
271     if(C1.c == C2.c) { assert(C1.r != C2.r); return
272     {};}
273     point vec = C2.c - C1.c;
274     ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
275     ;
276     ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
277     C1.r*C1.r - p*p*d2;
278     if (sum*sum < d2 or dif*dif > d2) return {};
279     point mid = C1.c + vec*p, per = point(-vec.y, vec
280     .x) * sqrt(max((ld)0, h2) / d2);
281     if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
282     return {mid + per, mid - per};
283 }
284 // minimum circle cover O(n) amortizado
285 circle min_circle_cover(vp v){
286     random_shuffle(v.begin(), v.end());
287     circle ans;
288     int n = v.size();
289     for(int i=0;i<n;i++) if(!ans.inside(v[i])){
290         ans = circle(v[i]);
291     }
292 }

```

```

290         for(int j=0;j<i;j++) if(!ans.inside(v[j])){
291             ans = circle(v[i], v[j]);
292             for(int k=0;k<j;k++) if(!ans.inside(v[k])){
293                 ans = circle(v[i], v[j], v[k]);
294             }
295         }
296     }
297     return ans;
298 }

```

8 Algorithms

8.1 Lis

```

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

```

8.2 Delta-encoding

```

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

```

8.3 Subsets

```

1 void subsets(vector<int>& nums){
2     int n = nums.size();
3     int powSize = 1 << n;
4
5     for(int counter = 0; counter < powSize; counter++){

```

```

6         for(int j = 0; j < n; j++){
7             if((counter & (1LL << j)) != 0) {
8                 cout << nums[j] << ' ';
9             }
10        }
11        cout << '\n';
12    }
13 }

```

8.4 Binary Search Last True

```

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

```

8.5 Ternary Search

```

1 double ternary_search(double l, double r) {
2     double eps = 1e-9; //set the error
3     limit here
4     while (r - l > eps) {
5         double m1 = l + (r - l) / 3;
6         double m2 = r - (r - l) / 3;
7         double f1 = f(m1); //evaluates the
8         function at m1
9         double f2 = f(m2); //evaluates the
10        function at m2
11        if (f1 < f2)
12            l = m1;
13        else
14            r = m2;
15    }
16    return f(l); //return the
17    maximum of f(x) in [l, r]
18 }

```

8.6 Binary Search First True

```

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

```

8.7 Biggest K

```

1 // Description: Gets sum of k biggest or k smallest
2 // elements in an array
3 // Problem: https://atcoder.jp/contests/abc306/tasks/
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.size()){
31             auto p = (prev(mt.mt.end()));
32             gt.add(*p);
33             mt.pop(*p);
34         }
35         while((int)mt.mt.size() && (int)gt.mt.size()
&&
36             *(gt.mt.begin()) < *(prev(mt.mt.end())) ){
37             ll u = *(gt.mt.begin());
38             ll v = *(prev(mt.mt.end()));
39             gt.pop(u); mt.pop(v);
40             gt.add(v); mt.add(u);
41         }
42     }
43     void add(ll x){
44         mt.add(x);
45         balancear();
46     }
47     void rem(ll x){
48         //x = -x;
49         if(mt.pop(x) == 0)
50             gt.pop(x);
51         balancear();
52     }
53 };
54
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(NULL);
58
59     int n, k, q; cin >> n >> k >> q;
60
61     BigK big = BigK(k);
62
63     int arr[n] = {};
64
65     while (q--) {
66         int pos, num; cin >> pos >> num;
67         pos--;
68         big.rem(arr[pos]);
69         arr[pos] = num;
70         big.add(arr[pos]);
71
72         cout << big.gt.s << '\n';
73     }
74
75     return 0;
76 }

```

9 Data Structures

9.1 Ordered Set

```

1 // Description:
2 // insert(k) - add element k to the ordered set
3 // erase(k) - remove element k from the ordered set
4 // erase(it) - remove element it points to from the
   ordered set
5 // order_of_key(k) - returns number of elements
   strictly smaller than k
6 // find_by_order(n) - return an iterator pointing to
   the k-th element in the ordered set (counting
   from zero).
7
8 // Problem:
9 // https://cses.fi/problemset/task/2169/
10
11 // Complexity:
12 // O(log n) for all operations
13
14 // How to use:
15 // ordered_set<int> os;
16 // cout << os.order_of_key(1) << '\n';
17 // cout << os.find_by_order(1) << '\n';
18
19 // Notes
20 // The ordered set only contains different elements
21 // By using less_equal<T> instead of less<T> on using
   ordered_set declaration
22 // The ordered_set becomes an ordered_multiset
23 // So the set can contain elements that are equal
24
25 #include <ext/pb_ds/assoc_container.hpp>
26 #include <ext/pb_ds/tree_policy.hpp>
27
28 using namespace __gnu_pbds;
29 template <typename T>
30 using ordered_set = tree<T, null_type, less<T>,
   rb_tree_tag, tree_order_statistics_node_update>;
31
32 void Erase(ordered_set<int>& a, int x){
33     int r = a.order_of_key(x);
34     auto it = a.find_by_order(r);
35     a.erase(it);
36 }

```

9.2 Priority Queue

```

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

```

9.3 Dsu

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX = 1e6+17;
6
7 struct DSU {
8     int n;
9     vector<int> link, sizes;
10
11     DSU(int n) {
12         this->n = n;
13         link.assign(n+1, 0);
14         sizes.assign(n+1, 1);
15
16         for (int i = 0; i <= n; i++)
17             link[i] = i;
18     }
19
20     int find(int x) {
21         while (x != link[x])
22             x = link[x];
23
24         return x;
25     }
26
27     bool same(int a, int b) {
28         return find(a) == find(b);
29     }
30
31     void unite(int a, int b) {
32         a = find(a);
33         b = find(b);
34
35         if (a == b) return;
36
37         if (sizes[a] < sizes[b])
38             swap(a, b);
39
40         sizes[a] += sizes[b];
41         link[b] = a;
42     }
43
44     int size(int x) {
45         return sizes[x];
46     }
47 };
48
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(NULL);
52
53     int cities, roads; cin >> cities >> roads;
54     vector<int> final_roads;
55     int ans = 0;
56     DSU dsu = DSU(cities);
57     for (int i = 0, a, b; i < roads; i++) {
58         cin >> a >> b;
59         dsu.unite(a, b);
60     }
61
62     for (int i = 2; i <= cities; i++) {
63         if (!dsu.same(1, i)) {
64             ans++;
65             final_roads.push_back(i);
66             dsu.unite(1, i);
67         }
68     }
69
70     cout << ans << '\n';
71     for (auto e : final_roads) {
```

```
72         cout << "1 " << e << '\n';
73     }
74
75 }
```

9.4 Two Sets

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



```

60 }
61
62 bool rem(int x) {
63     n--;
64     auto it1 = small.find(x);
65     auto it2 = big.find(x);
66     bool flag = false;
67     if (it1 != small.end()) {
68         sums -= *it1;
69         small.erase(it1);
70         flag = true;
71     } else if (it2 != big.end()) {
72         sumb -= *it2;
73         big.erase(it2);
74         flag = true;
75     }
76     balance();
77     return flag;
78 }
79
80 ll sum_small() {
81     return sums;
82 }
83
84 ll sum_big() {
85     return sumb;
86 }
87
88 int median() {
89     return *big.begin();
90 }
91 };

```

9.5 Dynamic Implicit Sparse

```

1 // Description:
2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
5 // stored in an array
6 // and the queries need to be answered online so we
7 // can't sort the nodes and compress them
8 // we create nodes only when they are needed so there
9 // 'll be (Q*log(MAX)) nodes
10 // where Q is the number of queries and MAX is the
11 // maximum index a node can assume
12
13 // Query - get sum of elements from range (l, r)
14 // inclusive
15 // Update - update element at position id to a value
16 // val
17
18 // Problem:
19 // https://cses.fi/problemset/task/1648
20
21 // Complexity:
22 // O(log n) for both query and update
23
24 // How to use:
25 // MAX is the maximum index a node can assume
26
27 // Segtree seg = Segtree(MAX);
28
29 typedef long long ftype;
30
31 const int MAX = 1e9+17;
32
33 struct Segtree {
34     vector<ftype> seg, d, e;
35     const ftype NEUTRAL = 0;
36     int n;
37
38     Segtree(int n) {

```

```

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
68         return;
69     }
70
71     int m = (ini + fim) >> 1;
72
73     if (id <= m) {
74         if (e[pos] == 0) e[pos] = create();
75         update(e[pos], ini, m, id, val);
76     } else {
77         if (d[pos] == 0) d[pos] = create();
78         update(d[pos], m + 1, fim, id, val);
79     }
80
81     seg[pos] = f(seg[e[pos]], seg[d[pos]]);
82 }
83
84 ftype query(int p, int q) {
85     return query(1, 1, n, p, q);
86 }
87
88 void update(int id, int val) {
89     update(1, 1, n, id, val);
90 }
91 };

```

9.6 Segtree2d

```

1 // Description:
2 // Indexed at zero
3 // Given a N x M grid, where i represents the row and
4 // j the column, perform the following operations
5 // update(j, i) - update the value of grid[i][j]
6 // query(j1, j2, i1, i2) - return the sum of values
7 // inside the rectangle
8 // defined by grid[i1][j1] and grid[i2][j2] inclusive
9
10 // Problem:
11 // https://cses.fi/problemset/task/1739/

```

```

10
11 // Complexity:
12 // Time complexity:
13 // O(log N * log M) for both query and update
14 // O(N * M) for build
15 // Memory complexity:
16 // 4 * M * N
17
18 // How to use:
19 // Segtree2D seg = Segtree2D(n, n);
20 // vector<vector<int>> v(n, vector<int>(n));
21 // seg.build(v);
22
23 // Notes
24 // Indexed at zero
25
26 struct Segtree2D {
27     const int MAXN = 1025;
28     int N, M;
29
30     vector<vector<int>> seg;
31
32     Segtree2D(int N, int M) {
33         this->N = N;
34         this->M = M;
35         seg.resize(2*MAXN, vector<int>(2*MAXN));
36     }
37
38     void buildY(int noX, int lX, int rX, int noY, int
        lY, int rY, vector<vector<int>> &v){
39         if(lY == rY){
40             if(lX == rX){
41                 seg[noX][noY] = v[rX][rY];
42             }else{
43                 seg[noX][noY] = seg[2*noX+1][noY] +
44                 seg[2*noX+2][noY];
45             }
46         }else{
47             int m = (lY+rY)/2;
48
49             buildY(noX, lX, rX, 2*noY+1, lY, m, v);
50             buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);
51
52             seg[noX][noY] = seg[noX][2*noY+1] + seg[
53             noX][2*noY+2];
54         }
55     }
56
57     void buildX(int noX, int lX, int rX, vector<
58     vector<int>> &v){
59         if(lX != rX){
60             int m = (lX+rX)/2;
61
62             buildX(2*noX+1, lX, m, v);
63             buildX(2*noX+2, m+1, rX, v);
64         }
65
66         buildY(noX, lX, rX, 0, 0, M - 1, v);
67     }
68
69     void updateY(int noX, int lX, int rX, int noY,
70     int lY, int rY, int y){
71         if(lY == rY){
72             if(lX == rX){
73                 seg[noX][noY] = !seg[noX][noY];
74             }else{
75                 seg[noX][noY] = seg[2*noX+1][noY] +
76                 seg[2*noX+2][noY];
77             }
78         }else{
79             int m = (lY+rY)/2;
80
81             if(y <= m){
82                 updateY(noX, lX, rX, 2*noY+1, lY, m, y
83                 );
84             }else if(m < y){
85                 updateY(noX, lX, rX, 2*noY+2, m+1, rY
86                 , y);
87             }
88
89             seg[noX][noY] = seg[noX][2*noY+1] + seg[
90             noX][2*noY+2];
91         }
92     }
93
94     void updateX(int noX, int lX, int rX, int x, int
95     y){
96         int m = (lX+rX)/2;
97
98         if(lX != rX){
99             if(x <= m){
100                 updateX(2*noX+1, lX, m, x, y);
101             }else if(m < x){
102                 updateX(2*noX+2, m+1, rX, x, y);
103             }
104         }
105
106         updateY(noX, lX, rX, 0, 0, M - 1, y);
107     }
108
109     int queryY(int noX, int noY, int lY, int rY, int
110     aY, int bY){
111         if(aY <= lY && rY <= bY) return seg[noX][noY
112         ];
113
114         int m = (lY+rY)/2;
115
116         if(bY <= m) return queryY(noX, 2*noY+1, lY, m
117         , aY, bY);
118         if(m < aY) return queryY(noX, 2*noY+2, m+1,
119         rY, aY, bY);
120
121         return queryY(noX, 2*noY+1, lY, m, aY, bY) +
122         queryY(noX, 2*noY+2, m+1, rY, aY, bY);
123     }
124
125     int queryX(int noX, int lX, int rX, int aX, int
126     bX, int aY, int bY){
127         if(aX <= lX && rX <= bX) return queryY(noX,
128         0, 0, M - 1, aY, bY);
129
130         int m = (lX+rX)/2;
131
132         if(bX <= m) return queryX(2*noX+1, lX, m, aX,
133         bX, aY, bY);
134         if(m < aX) return queryX(2*noX+2, m+1, rX, aX
135         , bX, aY, bY);
136
137         return queryX(2*noX+1, lX, m, aX, bX, aY, bY)
138         + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
139     }
140
141     void build(vector<vector<int>> &v) {
142         buildX(0, 0, N - 1, v);
143     }
144
145     int query(int aX, int bX, int aY, int bY) {
146         return queryX(0, 0, N - 1, aX, bX, aY, bY);
147     }
148
149     void update(int x, int y) {
150         updateX(0, 0, N - 1, x, y);
151     }
152 }

```

9.7 Minimum And Amount

```
1 // Description:
2 // Query - get minimum element in a range (l, r)
   inclusive
3 // and also the number of times it appears in that
   range
4 // Update - update element at position id to a value
   val
5
6 // Problem:
7 // https://codeforces.com/edu/course/2/lesson/4/1/
   practice/contest/273169/problem/C
8
9 // Complexity:
10 // O(log n) for both query and update
11
12 // How to use:
13 // Segtree seg = Segtree(n);
14 // seg.build(v);
15
16 #define pii pair<int, int>
17 #define mp make_pair
18 #define ff first
19 #define ss second
20
21 const int INF = 1e9+17;
22
23 typedef pii ftype;
24
25 struct Segtree {
26     vector<ftype> seg;
27     int n;
28     const ftype NEUTRAL = mp(INF, 0);
29
30     Segtree(int n) {
31         int sz = 1;
32         while (sz < n) sz *= 2;
33         this->n = sz;
34
35         seg.assign(2*sz, NEUTRAL);
36     }
37
38     ftype f(ftype a, ftype b) {
39         if (a.ff < b.ff) return a;
40         if (b.ff < a.ff) return b;
41
42         return mp(a.ff, a.ss + b.ss);
43     }
44
45     ftype query(int pos, int ini, int fim, int p, int
46     q) {
47         if (ini >= p && fim <= q) {
48             return seg[pos];
49         }
50
51         if (q < ini || p > fim) {
52             return NEUTRAL;
53         }
54
55         int e = 2*pos + 1;
56         int d = 2*pos + 2;
57         int m = ini + (fim - ini) / 2;
58
59         return f(query(e, ini, m, p, q), query(d, m +
60         1, fim, p, q));
61
62     }
63
64     void update(int pos, int ini, int fim, int id,
65     int val) {
66         if (ini > id || fim < id) {
67             return;
68         }
69     }
```

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

9.8 Lazy Addition To Segment

```
1 // Description:
2 // Query - get sum of elements from range (l, r)
   inclusive
3 // Update - add a value val to elementos from range (
   l, r) inclusive
4
5 // Problem:
6 // https://codeforces.com/edu/course/2/lesson/5/1/
   practice/contest/279634/problem/A
7
8 // Complexity:
9 // O(log n) for both query and update
10
11 // How to use:
12 // Segtree seg = Segtree(n);
```

```

13 // seg.build(v);
14
15 // Notes
16 // Change neutral element and f function to perform a
    different operation
17
18 const long long INF = 1e18+10;
19
20 typedef long long ftype;
21
22 struct Segtree {
23     vector<ftype> seg;
24     vector<ftype> lazy;
25     int n;
26     const ftype NEUTRAL = 0;
27     const ftype NEUTRAL_LAZY = -1; // change to -INF
    if there are negative numbers
28
29     Segtree(int n) {
30         int sz = 1;
31         while (sz < n) sz *= 2;
32         this->n = sz;
33
34         seg.assign(2*sz, NEUTRAL);
35         lazy.assign(2*sz, NEUTRAL_LAZY);
36     }
37
38     ftype apply_lazy(ftype a, ftype b, int len) {
39         if (b == NEUTRAL_LAZY) return a;
40         if (a == NEUTRAL_LAZY) return b * len;
41         else return a + b * len;
42     }
43
44     void propagate(int pos, int ini, int fim) {
45         if (ini == fim) {
46             return;
47         }
48
49         int e = 2*pos + 1;
50         int d = 2*pos + 2;
51         int m = ini + (fim - ini) / 2;
52
53         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
54         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
55
56         seg[e] = apply_lazy(seg[e], lazy[pos], m -
ini + 1);
57         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
m);
58
59         lazy[pos] = NEUTRAL_LAZY;
60     }
61
62     ftype f(ftype a, ftype b) {
63         return a + b;
64     }
65
66     ftype query(int pos, int ini, int fim, int p, int
q) {
67         propagate(pos, ini, fim);
68
69         if (ini >= p && fim <= q) {
70             return seg[pos];
71         }
72
73         if (q < ini || p > fim) {
74             return NEUTRAL;
75         }
76
77         int e = 2*pos + 1;
78         int d = 2*pos + 2;
79         int m = ini + (fim - ini) / 2;
80
81         return f(query(e, ini, m, p, q), query(d, m +
1, fim, p, q));
82     }
83
84     void update(int pos, int ini, int fim, int p, int
q, int val) {
85         propagate(pos, ini, fim);
86
87         if (ini > q || fim < p) {
88             return;
89         }
90
91         if (ini >= p && fim <= q) {
92             lazy[pos] = apply_lazy(lazy[pos], val, 1)
;
93             seg[pos] = apply_lazy(seg[pos], val, fim
- ini + 1);
94
95             return;
96         }
97
98         int e = 2*pos + 1;
99         int d = 2*pos + 2;
100         int m = ini + (fim - ini) / 2;
101
102         update(e, ini, m, p, q, val);
103         update(d, m + 1, fim, p, q, val);
104
105         seg[pos] = f(seg[e], seg[d]);
106     }
107
108     void build(int pos, int ini, int fim, vector<int>
&v) {
109         if (ini == fim) {
110             if (ini < (int)v.size()) {
111                 seg[pos] = v[ini];
112             }
113             return;
114         }
115
116         int e = 2*pos + 1;
117         int d = 2*pos + 2;
118         int m = ini + (fim - ini) / 2;
119
120         build(e, ini, m, v);
121         build(d, m + 1, fim, v);
122
123         seg[pos] = f(seg[e], seg[d]);
124     }
125
126     ftype query(int p, int q) {
127         return query(0, 0, n - 1, p, q);
128     }
129
130     void update(int p, int q, int val) {
131         update(0, 0, n - 1, p, q, val);
132     }
133
134     void build(vector<int> &v) {
135         build(0, 0, n - 1, v);
136     }
137
138     void debug() {
139         for (auto e : seg) {
140             cout << e << ' ';
141         }
142         cout << '\n';
143         for (auto e : lazy) {
144             cout << e << ' ';
145         }
146         cout << '\n';
147         cout << '\n';
148     }

```

```
149 };
```

9.9 Segment With Maximum Sum

```
1 // Description:
2 // Query - get sum of segment that is maximum among
  all segments
3 // E.g
4 // Array: 5 -4 4 3 -5
5 // Maximum segment sum: 8 because 5 + (-4) + 4 = 8
6 // Update - update element at position id to a value
  val
7
8 // Problem:
9 // https://codeforces.com/edu/course/2/lesson/4/2/
  practice/contest/273278/problem/A
10
11 // Complexity:
12 // O(log n) for both query and update
13
14 // How to use:
15 // Segtree seg = Segtree(n);
16 // seg.build(v);
17
18 // Notes
19 // The maximum segment sum can be a negative number
20 // In that case, taking zero elements is the best
  choice
21 // So we need to take the maximum between 0 and the
  query
22 // max(OLL, seg.query(0, n).max_seg)
23
24 using ll = long long;
25
26 typedef ll ftype_node;
27
28 struct Node {
29     ftype_node max_seg;
30     ftype_node pref;
31     ftype_node suf;
32     ftype_node sum;
33
34     Node(ftype_node max_seg, ftype_node pref,
35         ftype_node suf, ftype_node sum) : max_seg(max_seg),
36         pref(pref), suf(suf), sum(sum) {};
37 };
38
39 typedef Node ftype;
40
41 struct Segtree {
42     vector<ftype> seg;
43     int n;
44     const ftype NEUTRAL = Node(0, 0, 0, 0);
45
46     Segtree(int n) {
47         int sz = 1;
48         // potencia de dois mais proxima
49         while (sz < n) sz *= 2;
50         this->n = sz;
51
52         // numero de nos da seg
53         seg.assign(2*sz, NEUTRAL);
54
55         ftype f(ftype a, ftype b) {
56             ftype_node max_seg = max({a.max_seg, b.
57 max_seg, a.suf + b.pref});
58             ftype_node pref = max(a.pref, a.sum + b.pref);
59
60             ftype_node suf = max(b.suf, b.sum + a.suf);
61             ftype_node sum = a.sum + b.sum;
62
63             return Node(max_seg, pref, suf, sum);
64         }
65     }
66 }
```

```
61 }
62
63 ftype query(int pos, int ini, int fim, int p, int
64 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 +
78 1, fim, p, q));
79 }
80
81 void update(int pos, int ini, int fim, int id,
82 int val) {
83     if (ini > id || fim < id) {
84         return;
85     }
86
87     if (ini == id && fim == id) {
88         seg[pos] = Node(val, val, val, val);
89     }
90
91     return;
92
93     int e = 2*pos + 1;
94     int d = 2*pos + 2;
95     int m = ini + (fim - ini) / 2;
96
97     update(e, ini, m, id, val);
98     update(d, m + 1, fim, id, val);
99
100     seg[pos] = f(seg[e], seg[d]);
101 }
102
103 void build(int pos, int ini, int fim, vector<int>
104 &v) {
105     if (ini == fim) {
106         // se a çãposio existir no array original
107         // seg tamanho potencia de dois
108         if (ini < (int)v.size()) {
109             seg[pos] = Node(v[ini], v[ini], v[ini],
110 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 id, int val) {
130     update(0, 0, n - 1, id, val);
131 }
132
133 void build(vector<int> &v) {
134 }
```

```

129     build(0, 0, n - 1, v);
130 }
131
132 void debug() {
133     for (auto e : seg) {
134         cout << e.max_seg << ' ' << e.pref << ' '
135         << e.suf << ' ' << e.sum << '\n';
136     }
137     cout << '\n';
138 };

```

9.10 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 // O(log n) for both query and update
12 // How to use:
13 // Segtree seg = Segtree(n);
14 // seg.build(v);
15
16 // Notes
17 // Change neutral element and f function to perform a
18 // different operation
19 // If you want to change the operations to point
20 // query and range update
21 // Use the same segtree, but perform the following
22 // operations
23 // Query - seg.query(0, id);
24 // Update - seg.update(l, v); seg.update(r + 1, -v);
25
26 typedef long long ftype;
27
28 struct Segtree {
29     vector<ftype> seg;
30     int n;
31     const ftype NEUTRAL = 0;
32
33     Segtree(int n) {
34         int sz = 1;
35         while (sz < n) sz *= 2;
36         this->n = sz;
37
38         seg.assign(2*sz, NEUTRAL);
39     }
40
41     ftype f(ftype a, ftype b) {
42         return a + b;
43     }
44
45     ftype query(int pos, int ini, int fim, int p, int
46     q) {
47         if (ini >= p && fim <= q) {
48             return seg[pos];
49         }
50
51         if (q < ini || p > fim) {
52             return NEUTRAL;
53         }
54
55         int e = 2*pos + 1;

```

```

53     int d = 2*pos + 2;
54     int m = ini + (fim - ini) / 2;
55
56     return f(query(e, ini, m, p, q), query(d, m +
57     1, fim, p, q));
58 }
59
60 void update(int pos, int ini, int fim, int id,
61 int val) {
62     if (ini > id || fim < id) {
63         return;
64     }
65
66     if (ini == id && fim == id) {
67         seg[pos] = val;
68     }
69
70     return;
71
72     int e = 2*pos + 1;
73     int d = 2*pos + 2;
74     int m = ini + (fim - ini) / 2;
75
76     update(e, ini, m, id, val);
77     update(d, m + 1, fim, id, val);
78
79     seg[pos] = f(seg[e], seg[d]);
80 }
81
82 void build(int pos, int ini, int fim, vector<int>
83 &v) {
84     if (ini == fim) {
85         if (ini < (int)v.size()) {
86             seg[pos] = v[ini];
87         }
88         return;
89     }
90
91     int e = 2*pos + 1;
92     int d = 2*pos + 2;
93     int m = ini + (fim - ini) / 2;
94
95     build(e, ini, m, v);
96     build(d, m + 1, fim, v);
97
98     seg[pos] = f(seg[e], seg[d]);
99 }
100
101 ftype query(int p, int q) {
102     return query(0, 0, n - 1, p, q);
103 }
104
105 void update(int id, int val) {
106     update(0, 0, n - 1, id, val);
107 }
108
109 void build(vector<int> &v) {
110     build(0, 0, n - 1, v);
111 }
112
113 void debug() {
114     for (auto e : seg) {
115         cout << e << ' ';
116     }
117     cout << '\n';
118 }
119 };

```

9.11 Lazy Assignment To Segment

```

1 const long long INF = 1e18+10;
2
3 typedef long long ftype;

```

```

4
5 struct Segtree {
6     vector<ftype> seg;
7     vector<ftype> lazy;
8     int n;
9     const ftype NEUTRAL = 0;
10    const ftype NEUTRAL_LAZY = -1; // Change to -INF
    if there are negative numbers
11
12    Segtree(int n) {
13        int sz = 1;
14        // potencia de dois mais proxima
15        while (sz < n) sz *= 2;
16        this->n = sz;
17
18        // numero de nos da seg
19        seg.assign(2*sz, NEUTRAL);
20        lazy.assign(2*sz, NEUTRAL_LAZY);
21    }
22
23    ftype apply_lazy(ftype a, ftype b, int len) {
24        if (b == NEUTRAL_LAZY) return a;
25        if (a == NEUTRAL_LAZY) return b * len;
26        else return b * len;
27    }
28
29    void propagate(int pos, int ini, int fim) {
30        if (ini == fim) {
31            return;
32        }
33
34        int e = 2*pos + 1;
35        int d = 2*pos + 2;
36        int m = ini + (fim - ini) / 2;
37
38        lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
39        lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
40
41        seg[e] = apply_lazy(seg[e], lazy[pos], m -
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        }
60
61        if (q < ini || p > fim) {
62            return NEUTRAL;
63        }
64
65        int e = 2*pos + 1;
66        int d = 2*pos + 2;
67        int m = ini + (fim - ini) / 2;
68
69        return f(query(e, ini, m, p, q), query(d, m
70        + 1, fim, p, q));
71    }
72
73    void update(int pos, int ini, int fim, int p, int
74    q, int val) {
75        propagate(pos, ini, fim);
76
77        if (ini > q || fim < p) {
78            return;
79        }
80
81        if (ini >= p && fim <= q) {
82            lazy[pos] = apply_lazy(lazy[pos], val, 1)
83            ;
84            seg[pos] = apply_lazy(seg[pos], val, fim
85            - ini + 1);
86
87            return;
88        }
89
90        int e = 2*pos + 1;
91        int d = 2*pos + 2;
92        int m = ini + (fim - ini) / 2;
93
94        update(e, ini, m, p, q, val);
95        update(d, m + 1, fim, p, q, val);
96
97        seg[pos] = f(seg[e], seg[d]);
98    }
99
100    void build(int pos, int ini, int fim, vector<int>
101    &v) {
102        if (ini == fim) {
103            // se a posição existir no array original
104            // seg tamanho potencia de dois
105            if (ini < (int)v.size()) {
106                seg[pos] = 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 p, int q, int val) {
126        update(0, 0, n - 1, p, q, 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 << ' ';
136        }
137        cout << '\n';
138        for (auto e : lazy) {
139            cout << e << ' ';
140        }
141        cout << '\n';
142        cout << '\n';
143    }
144
145    1 // Description:

```

9.12 Lazy Dynamic Implicit Sparse

```

2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
5 // and the queries need to be answered online so we
6 // we create nodes only when they are needed so there
7 // where Q is the number of queries and MAX is the
8 // maximum index a node can assume
9 // Query - get sum of elements from range (l, r)
10 // Update - update element at position id to a value
11 // Problem:
12 // https://oj.uz/problem/view/IZh012_apple
13 // Complexity:
14 // O(log n) for both query and update
15 // How to use:
16 // MAX is the maximum index a node can assume
17 // Create a default null node
18 // Create a node to be the root of the segtree
19 // Segtree seg = Segtree(MAX);
20
21 const int MAX = 1e9+10;
22 const long long INF = 1e18+10;
23
24 typedef long long ftype;
25
26 struct Segtree {
27     vector<ftype> seg, d, e, lazy;
28     const ftype NEUTRAL = 0;
29     const ftype NEUTRAL_LAZY = -1; // change to -INF
30     if the elements can be negative
31     int n;
32
33     Segtree(int n) {
34         this->n = n;
35         create();
36         create();
37     }
38
39     ftype apply_lazy(ftype a, ftype b, int len) {
40         if (b == NEUTRAL_LAZY) return a;
41         else return b * len; // change to a + b * len
42         to add to an element instead of updating it
43     }
44
45     void propagate(int pos, int ini, int fim) {
46         if (seg[pos] == 0) return;
47
48         if (ini == fim) {
49             return;
50         }
51
52         int m = (ini + fim) >> 1;
53
54         if (e[pos] == 0) e[pos] = create();
55         if (d[pos] == 0) d[pos] = create();
56
57         lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[
58 pos], 1);
59         lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[
60 pos], 1);
61
62         seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[
63 pos], m - ini + 1);
64         seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[
65 pos], fim - m);
66
67         lazy[pos] = NEUTRAL_LAZY;
68     }
69
70     ftype f(ftype a, ftype b) {
71         return a + b;
72     }
73
74     ftype create() {
75         seg.push_back(0);
76         e.push_back(0);
77         d.push_back(0);
78         lazy.push_back(-1);
79         return seg.size() - 1;
80     }
81
82     ftype query(int pos, int ini, int fim, int p, int
83 q) {
84         propagate(pos, ini, fim);
85         if (q < ini || p > fim) return NEUTRAL;
86         if (pos == 0) return 0;
87         if (p <= ini && fim <= q) return seg[pos];
88         int m = (ini + fim) >> 1;
89         return f(query(e[pos], ini, m, p, q), query(d
90 [pos], m + 1, fim, p, q));
91     }
92
93     void update(int pos, int ini, int fim, int p, int
94 q, int val) {
95         propagate(pos, ini, fim);
96         if (ini > q || fim < p) {
97             return;
98         }
99
100         if (ini >= p && fim <= q) {
101             lazy[pos] = apply_lazy(lazy[pos], val, 1)
102 ;
103             seg[pos] = apply_lazy(seg[pos], val, fim
104 - ini + 1);
105         }
106
107         return;
108
109         int m = (ini + fim) >> 1;
110
111         if (e[pos] == 0) e[pos] = create();
112         update(e[pos], ini, m, p, q, val);
113
114         if (d[pos] == 0) d[pos] = create();
115         update(d[pos], m + 1, fim, p, q, val);
116
117         seg[pos] = f(seg[e[pos]], seg[d[pos]]);
118     }
119
120     ftype query(int p, int q) {
121         return query(1, 1, n, p, q);
122     }
123
124     void update(int p, int q, int val) {
125         update(1, 1, n, p, q, val);
126     }
127 };

```

9.13 Persistent

```

1 // Description:
2 // Persistent segtree allows for you to save the
3 // different versions of the segtree between each
4 // update
5 // Indexed at one
6 // Query - get sum of elements from range (l, r)
7 // inclusive

```



```

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
59         seg[novo] = seg[pos];
60         e[novo] = e[pos];
61         d[novo] = d[pos];
62
63         if (ini == fim) {
64             seg[novo] = val;
65             return novo;
66         }
67
68         int m = (ini + fim) >> 1;
69
70         if (id <= m) e[novo] = update(e[novo], ini, m
    , id, val);
71         else d[novo] = update(d[novo], m + 1, fim, id
    , val);
72
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 };

```