Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

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

1.3 Subsets

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

1.4 Matrix Exponentiation

```
1 // Description:
2 // Calculate the nth term of a linear recursion
3
4 // Example Fibonacci:
5 // Given a linear recurrence, for example fibonacci
6 // F(n) = n, x <= 1
7 // F(n) = F(n - 1) + F(n - 2), x > 1
8
9 // The recurrence has two terms, so we can build a
    matrix 2 x 1 so that
10 // n + 1 = transition * n
11
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
    distinct colors so that
18 // no adjacent place has the same color. In how many
     different ways can you do that?
19 // There are 6 ways for the first column to be
    colored using 3 distinct colors
20 // ans 6 ways using 2 equal colors and 1 distinct one
21
22 // Adding another column, there are:
23 // 3 ways to go from 2 equal to 2 equal
24 // 2 ways to go from 2 equal to 3 distinct
25 // 2 ways to go from 3 distinct to 2 equal
26 // 2 ways to go from 3 distinct to 3 distinct
27
28 // So we star with matrix 6 6 and multiply it by the
     transition 3 2 and get 18 12
29 //              6 6
                2 2        12 12
30 // the we can exponentiate this matrix to find the
    nth column
31
32 // Problem:
33 // https://cses.fi/problemset/task/1722/
34
35 // Complexity:
36 // O(log n)
37
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>>& matrix){
56         mat = matrix;
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                 mat[i][i] = 1;
67             }
68         }
69     }
70
71     Matriz operator * (Matriz a) {
72         assert(columns == a.rows);
73         vector<vector<ll>> resp(rows, vector<ll>(a.
    columns, 0));
74
75         for(int i = 0; i < rows; i++){
76             for(int j = 0; j < a.columns; j++){
77                 for(int k = 0; k < a.rows; k++){
78                     resp[i][j] = (resp[i][j] + (mat[i
    ][k] * 1LL * a[k][j]) % MOD) % MOD;
79                 }
80             }
81         }
82         return Matriz(resp);
83     }
84
85     Matriz operator + (Matriz a) {
86         assert(rows == a.rows && columns == a.columns
    );
```

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

1.5 Crt

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

1.6 Binary To Decimal

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

1.7 Fast Exponentiation

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

1.8 Linear Diophantine Equation

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

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

1.9 Function Root

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

1.10 Sieve Of Eratosthenes

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

1.11 Horner Algorithm

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

1.12 Multiplicative Inverse

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

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

1.13   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  }
```

1.14   Set Operations

```
1  // Complexity;
2  // O(n * m) being n and m the sizes of the two sets
3  // 2*(count1+count2)-1 (where countX is the distance
      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()));
```

1.15   Divisors

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

1.16   Check If Bit Is On

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

1.17   Prime Factors

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

2   DP

2.1   Knapsack With Index

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

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

2.2 Substr Palindrome

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

2.3 Edit Distance

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

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

2.4 Knapsack

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

2.5 Digits

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

2.6 Coins

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

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

2.7   Minimum Coin Change

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

2.8   Kadane

```
1  // achar uma subsequencia continua no array que a
       soma seja a maior possivel
2  // nesse caso vc precisa multiplicar exatamente 1
       elemento da subsequencia
3  // e achar a maior soma com isso
4
5  int n, x, arr[MAX], tab[MAX][2]; // tab[maior
       resposta no intervalo][foi multiplicado ou ãno]
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);
```

```cpp
#include <bits/stdc++.h>
using namespace std;

#define pb push_back
#define mp make_pair
#define mt make_tuple
#define ff first
#define ss second
#define ld long double
#define ll long long
#define int long long
#define pii pair<int, int>
#define tii tuple<int, int, int>

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



    return 0;
}
```

4 Strings

4.1 Hash

```cpp
// Description:
// Turns a string into a integer.
// If the hash is different then the strings are
    different.
// If the hash is the same the strings may be
    different.

// Problem:
// https://codeforces.com/gym/104518/problem/I

// Complexity:
// O(n) to calculate the hash
// O(1) to query

// Notes:
// Primes 1000000007, 1000041323, 100663319,
    201326611, 1000015553, 1000028537

struct Hash {
    const ll P = 31;
    int n; string s;
    vector<ll> h, hi, p;
    Hash() {}
    Hash(string s): s(s), n(s.size()), h(n), hi(n), p
    (n) {
        for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1)
    % MOD;
        for (int i=0;i<n;i++)
            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
        for (int i=n-1;i>=0;i--)
            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
    % MOD;
    }
    int query(int l, int r) {
        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
    0));
        return hash < 0 ? hash + MOD : hash;
    }
    int query_inv(int l, int r) {
        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
    +1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
};
```

4.2 Kmp

```cpp
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

4.3 Generate All Permutations

```cpp
vector<string> generate_permutations(string s) {
    int n = s.size();
    vector<string> ans;

    sort(s.begin(), s.end());

    do {
        ans.push_back(s);
    } while (next_permutation(s.begin(), s.end()));

    return ans;
}
```

4.4 Generate All Sequences Length K

```cpp
// gera todas as ípossveis êsequncias usando as letras
    em set (de comprimento n) e que tenham tamanho k
// sequence = ""
vector<string> generate_sequences(char set[], string
    sequence, int n, int k) {
    if (k == 0){
        return { sequence };
    }

    vector<string> ans;
    for (int i = 0; i < n; i++) {
        auto aux = generate_sequences(set, sequence +
    set[i], n, k - 1);
        ans.insert(ans.end(), aux.begin(), aux.end())
    ;
        // for (auto e : aux) ans.push_back(e);
    }

    return ans;
}
```

4.5 Suffix Array

```cpp
// Description:
// Suffix array is an array with the indexes of the
    starting letter of every
// suffix in an array sorted in lexicographical order
    .

// Problem:
// https://codeforces.com/edu/course/2/lesson/2/1/
    practice/contest/269100/problem/A

// Complexity:
// O(n log n) with radix sort
// O(n log ^ 2 n) with regular sort

// Notes:
// Relevant Problems
```

```cpp
14  // Substring search: Queries to know whether a given
       substring is present in a string
15  // Binary search for the first suffix that is greater
       or equal
16  // O(log n |p|) where |p| is the total size of the
       substrings queried
17  //
18  // Substring size: Queries to know how many times a
       given substring appears in a string
19  // Binary search both for first and last that is
       greater or equal
20  //
21  // Number of different substrings:
22  // A given suffix gives sz new substrings being sz
       the size of the suffix
23  // We can subtract the lcp (longest common prefix) to
       remove substrings
24  // that were already counted.
25  //
26  // Longest common substring between two strings:
27  // We can calculate the suffix array and lcp array of
       the two strings
28  // concantened with a character greater than $ and
       smaller than A (like '&')
29  // The answer will be the lcp between two consecutive
       suffixes that belong to different strings
30  // (index at suffix array <= size of the first array)
31
32  void radix_sort(vector<pair<pair<int, int>, int>>& a)
       {
33    int n = a.size();
34    vector<pair<pair<int, int>, int>> ans(n);
35
36    vector<int> count(n);
37
38    for (int i = 0; i < n; i++) {
39      count[a[i].first.second]++;
40    }
41
42    vector<int> p(n);
43
44    p[0] = 0;
45    for (int i = 1; i < n; i++) {
46      p[i] = p[i - 1] + count[i - 1];
47    }
48
49    for (int i = 0; i < n; i++) {
50      ans[p[a[i].first.second]++] = a[i];
51    }
52
53    a = ans;
54
55    count.assign(n, 0);
56
57    for (int i = 0; i < n; i++) {
58      count[a[i].first.first]++;
59    }
60
61    p.assign(n, 0);
62
63    p[0] = 0;
64    for (int i = 1; i < n; i++) {
65      p[i] = p[i - 1] + count[i - 1];
66    }
67
68    for (int i = 0; i < n; i++) {
69      ans[p[a[i].first.first]++] = a[i];
70    }
71
72    a = ans;
73  }
74
75  vector<int> p, c;

76
77  vector<int> suffix_array(string s) {
78    int n = s.size();
79    vector<pair<char, int>> a(n);
80    p.assign(n, 0);
81    c.assign(n, 0);
82
83    for (int i = 0; i < n; i++) {
84      a[i] = mp(s[i], i);
85    }
86
87    sort(a.begin(), a.end());
88
89    for (int i = 0; i < n; i++) {
90      p[i] = a[i].second;
91    }
92
93    c[p[0]] = 0;
94    for (int i = 1; i < n; i++) {
95      if (a[i].first == a[i - 1].first) c[p[i]] = c[p[i
         - 1]];
96      else c[p[i]] = c[p[i - 1]] + 1;
97    }
98
99    int k = 0;
100   while ((1 << k) < n) {
101     vector<pair<pair<int, int>, int >> a(n);
102     for (int i = 0; i < n; i++) {
103       a[i] = mp(mp(c[i], c[(i + (1 << k)) % n]), i);
104     }
105
106     radix_sort(a);
107
108     for (int i = 0; i < n; i++) {
109       p[i] = a[i].second;
110     }
111
112     c[p[0]] = 0;
113     for (int i = 1; i < n; i++) {
114       if (a[i].first == a[i - 1].first) c[p[i]] = c[p
         [i - 1]];
115       else c[p[i]] = c[p[i - 1]] + 1;
116     }
117
118     k++;
119   }
120
121   /* for (int i = 0; i < n; i++) {
122     for (int j = p[i]; j < n; j++) {
123       cout << s[j];
124     }
125     cout << '\n';
126   } */
127
128   return p;
129 }
130
131 // the first suffix will alway be $ the (n - 1)th
       character in the string
132 vector<int> lcp_array(string s) {
133   int n = s.size();
134   vector<int> ans(n);
135   // minimum lcp
136   int k = 0;
137   for (int i = 0; i < n - 1; i++) {
138     // indice in the suffix array p of suffix
         starting in i
139     int pi = c[i];
140     // start index of the previous suffix in suffix
         array
141     int j = p[pi - 1];
142     while (s[i + k] == s[j + k]) k++;
143     ans[pi] = k;
```

```
144     k = max(k - 1, 0);
145   }
146
147   return ans;
148 }
```

4.6   Lcs

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

4.7   Trie

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

```
14    }
15 };
16
17 vector<Vertex> t(1);
18
19 void add_string(string const& s) {
20     int v = 0;
21     for (char ch : s) {
22         int c = ch - 'a';
23         if (t[v].next[c] == -1) {
24             t[v].next[c] = t.size();
25             t.emplace_back(v, ch);
26         }
27         v = t[v].next[c];
28     }
29     t[v].output = true;
30 }
31
32 int go(int v, char ch);
33
34 int get_link(int v) {
35     if (t[v].link == -1) {
36         if (v == 0 || t[v].p == 0)
37             t[v].link = 0;
38         else
39             t[v].link = go(get_link(t[v].p), t[v].pch
     );
40     }
41     return t[v].link;
42 }
43
44 int go(int v, char ch) {
45     int c = ch - 'a';
46     if (t[v].go[c] == -1) {
47         if (t[v].next[c] != -1)
48             t[v].go[c] = t[v].next[c];
49         else
50             t[v].go[c] = v == 0 ? 0 : go(get_link(v),
      ch);
51     }
52     return t[v].go[c];
53 }
```

4.8   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             ++z[i];
9         if (i + z[i] - 1 > r)
10            l = i, r = i + z[i] - 1;
11    }
12    return z;
13 }
```

5   Misc

5.1   Split

```
1 vector<string> split(string txt, char key = ' '){
2     vector<string> ans;
3
4     string palTemp = "";
5     for(int i = 0; i < txt.size(); i++){
6
7         if(txt[i] == key){
```

```
 8              if(palTemp.size() > 0){
 9                  ans.push_back(palTemp);
10                  palTemp = "";
11              }
12          } else{
13              palTemp += txt[i];
14          }
15
16      }
17
18      if(palTemp.size() > 0)
19          ans.push_back(palTemp);
20
21      return ans;
22 }
```

5.2 Int128

```
 1 __int128 read() {
 2     __int128 x = 0, f = 1;
 3     char ch = getchar();
 4     while (ch < '0' || ch > '9') {
 5         if (ch == '-') f = -1;
 6         ch = getchar();
 7     }
 8     while (ch >= '0' && ch <= '9') {
 9         x = x * 10 + ch - '0';
10         ch = getchar();
11     }
12     return x * f;
13 }
14 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
       is appointed the root of the tree,
 4 // each subtree has at most floor(n/2) nodes.
 5
 6 // Problem:
 7 // https://cses.fi/problemset/task/2079/
 8
 9 // Complexity:
10 // O(n)
11
12 // How to use:
13 // get_subtree_size(0);
14 // cout << get_centroid(0) + 1 << endl;
15
16 int n;
17 vector<int> adj[MAX];
18 int subtree_size[MAX];
19
20 int get_subtree_size(int node, int par = -1) {
21   int &res = subtree_size[node];
22   res = 1;
23   for (int i : adj[node]) {
24     if (i == par) continue;
25     res += get_subtree_size(i, node);
26   }
```

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

6.2 Bipartite

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

6.3 Prim

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

```cpp
struct Edge {
    int w = INF, to = -1;
};

void prim() {
    int total_weight = 0;
    vector<bool> selected(n, false);
    vector<Edge> min_e(n);
    min_e[0].w = 0;

    for (int i=0; i<n; ++i) {
        int v = -1;
        for (int j = 0; j < n; ++j) {
            if (!selected[j] && (v == -1 || min_e[j].w < min_e[v].w))
                v = j;
        }

        if (min_e[v].w == INF) {
            cout << "No MST!" << endl;
            exit(0);
        }

        selected[v] = true;
        total_weight += min_e[v].w;
        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to << endl;

        for (int to = 0; to < n; ++to) {
            if (adj[v][to] < min_e[to].w)
                min_e[to] = {adj[v][to], v};
        }
    }

    cout << total_weight << endl;
}
```

6.4   Eulerian Undirected

```cpp
// Description:
// Hierholzer's Algorithm
// An Eulerian path is a path that passes through
    every edge exactly once.
// An Eulerian circuit is an Eulerian path that
    starts and ends on the same node.

// An Eulerian path exists in an undirected graph if
    the degree of every node is even (not counting
    self-edges)
// except for possibly exactly two nodes that have
    and odd degree (start and end nodes).
// An Eulerian circuit exists in an undirected graph
    if the degree of every node is even.

// The graph has to be conected (except for isolated
    nodes which are allowed because there
// are no edges connected to them).

// Problem:
// https://cses.fi/problemset/task/1691

// Complexity:
// O(E * log(E)) where E is the number of edges

// How to use
// Check whether the path exists before trying to
    find it
// Find the root - any node that has at least 1
    outgoing edge
// (if the problem requires that you start from a
    node v, the root will be the node v)
// Count the degree;
//
```

```cpp
// for (int i = 0; i < m; i++) {
//   int a, b; cin >> a >> b;
//   adj[a].pb(b); adj[b].pb(a);
//   root = a;
//   degree[a]++; degree[b]++;
// }

// Notes
// If you want to find a path start and ending nodes
    v and u
// if ((is_eulerian(n, root, start, end) != 1) || (
    start != v) || (end != u)) cout << "IMPOSSIBLE\n"
// It can be speed up to work on O(E) on average by
    using unordered_set instead of set

// It works when there are self loops, but not when
    there are multiple edges
// It the graph has multiple edges, add more notes to
    simulate the edges
// e.g
// 1 2
// 1 2
// 1 2
// becomes
// 3 4
// 4 1
// 1 2

vector<bool> visited;
vector<int> degree;
vector<vector<int>> adj;

void dfs(int v) {
  visited[v] = true;
  for (auto u : adj[v]) {
    if (!visited[u]) dfs(u);
  }
}

int is_eulerian(int n, int root, int& start, int& end
    ) {
  start = -1, end = -1;
  if (n == 1) return 2; // only one node
  visited.assign(n + 1, false);
  dfs(root);

  for (int i = 1; i <= n; i++) {
    if (!visited[i] && degree[i] > 0) return 0;
  }

  for (int i = 1; i <= n; i++) {
    if (start == -1 && degree[i] % 2 == 1) start = i;
    else if (end == -1 && degree[i] % 2 == 1) end = i
    ;
    else if (degree[i] % 2 == 1) return 0;
  }

  if (start == -1 && end == -1) {start = root; end =
    root; return 2;} // has eulerian circuit and path
  if (start != -1 && end != -1) return 1; // has
    eulerian path
  return 0; // no eulerian path nor circuit
}

vector<int> path;
vector<set<int>> mark;

void dfs_path(int v) {
  visited[v] = true;

  while (degree[v] != 0) {
    degree[v]--;
```

```
89      int u = adj[v][degree[v]];
90      if (mark[v].find(u) != mark[v].end()) continue;
91      mark[v].insert(u);
92      mark[u].insert(v);
93      int next_edge = adj[v][degree[v]];
94      dfs_path(next_edge);
95    }
96    path.pb(v);
97  }
98
99  void find_path(int n, int start) {
100   path.clear();
101   mark.resize(n + 1);
102   visited.assign(n + 1, false);
103   dfs_path(start);
104 }
```

6.5 Ford Fulkerson Edmonds Karp

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

```
49          }
50      }
51
52      return flow;
53 }
```

6.6 Hld Edge

```
1  // Description:
2  // Make queries and updates between two vertexes on a
       tree
3
4  // Problem:
5  // https://www.spoj.com/problems/QTREE/
6
7  // Complexity:
8  // O(log ^2 n) for both query and update
9
10 // How to use:
11 // HLD hld = HLD(n + 1, adj)
12
13 // Notes
14 // Change the root of the tree on the constructor if
       it's different from 1
15 // Use together with Segtree
16
17 struct HLD {
18   vector<int> parent;
19   vector<int> pos;
20   vector<int> head;
21   vector<int> subtree_size;
22   vector<int> level;
23   vector<int> heavy_child;
24   vector<ftype> subtree_weight;
25   vector<ftype> path_weight;
26   vector<vector<int>> adj;
27   vector<int> at;
28   Segtree seg = Segtree(0);
29   int cpos;
30   int n;
31   int root;
32
33   HLD() {}
34
35   HLD(int n, vector<vector<int>>& adj, int root = 1)
    : adj(adj), n(n), root(root) {
36     seg = Segtree(n);
37     cpos = 0;
38     at.assign(n, 0);
39     parent.assign(n, 0);
40     pos.assign(n, 0);
41     head.assign(n, 0);
42     subtree_size.assign(n, 1);
43     level.assign(n, 0);
44     heavy_child.assign(n, -1);
45     parent[root] = -1;
46     dfs(root, -1);
47     decompose(root, -1);
48   }
49
50   void dfs(int v, int p) {
51     parent[v] = p;
52     if (p != -1) level[v] = level[p] + 1;
53     for (auto u : adj[v]) {
54       if (u != p) {
55         dfs(u, v);
56         subtree_size[v] += subtree_size[u];
57         if (heavy_child[v] == -1 || subtree_size[u] >
    subtree_size[heavy_child[v]]) heavy_child[v] = u
    ;
58       }
59     }
60   }
```

```
61
62    void decompose(int v, int chead) {
63      // start a new path
64      if (chead == -1) chead = v;
65
66      // consecutive ids in the hld path
67      at[cpos] = v;
68      pos[v] = cpos++;
69      head[v] = chead;
70
71      // if not a leaf
72      if (heavy_child[v] != -1) decompose(heavy_child[v
      ], chead);
73
74      // light child
75      for (auto u : adj[v]){
76        // start new path
77        if (u != parent[v] && u != heavy_child[v])
      decompose(u, -1);
78      }
79    }
80
81    ll query_path(int a, int b) {
82      if (a == b) return 0;
83      if(pos[a] < pos[b]) swap(a, b);
84
85      if(head[a] == head[b]) return seg.query(pos[b] +
      1, pos[a]);
86      return seg.f(seg.query(pos[head[a]], pos[a]),
      query_path(parent[head[a]], b));
87    }
88
89    ftype query_subtree(int a) {
90      if (subtree_size[a] == 1) return 0;
91      return seg.query(pos[a] + 1, pos[a] +
      subtree_size[a] - 1);
92    }
93
94    void update_path(int a, int b, int x) {
95      if (a == b) return;
96      if(pos[a] < pos[b]) swap(a, b);
97
98      if(head[a] == head[b]) return (void)seg.update(
      pos[b] + 1, pos[a], x);
99      seg.update(pos[head[a]], pos[a], x); update_path(
      parent[head[a]], b, x);
100   }
101
102   void update_subtree(int a, int val) {
103     if (subtree_size[a] == 1) return;
104     seg.update(pos[a] + 1, pos[a] + subtree_size[a] -
       1, val);
105   }
106
107   // vertex
108   void update(int a, int val) {
109     seg.update(pos[a], pos[a], val);
110   }
111
112   //edge
113   void update(int a, int b, int val) {
114     if (parent[a] == b) swap(a, b);
115     update(b, val);
116   }
117
118   int lca(int a, int b) {
119     if(pos[a] < pos[b]) swap(a, b);
120     return head[a] == head[b] ? b : lca(parent[head[a
      ]], b);
121   }
122 };
```

6.7 Floyd Warshall

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  using ll = long long;
5
6  const int MAX  = 507;
7  const long long INF = 0x3f3f3f3f3f3f3f3fLL;
8
9  ll dist[MAX][MAX];
10 int n;
11
12 void floyd_warshall() {
13     for (int i = 0; i < n; i++) {
14         for (int j = 0; j < n; j++) {
15             if (i == j) dist[i][j] = 0;
16             else if (!dist[i][j]) dist[i][j] = INF;
17         }
18     }
19
20     for (int k = 0; k < n; k++) {
21         for (int i = 0; i < n; i++) {
22             for (int j = 0; j < n; j++) {
23                 // trata o caso no qual o grafo tem
      arestas com peso negativo
24                 if (dist[i][k] < INF && dist[k][j] <
      INF){
25                     dist[i][j] = min(dist[i][j], dist
      [i][k] + dist[k][j]);
26                 }
27             }
28         }
29     }
30 }
```

6.8 Lca

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

```
32    visited[1] = true;
33
34    while (!q.empty()) {
35      auto [v, depth] = q.front();
36      q.pop();
37      level[v] = depth;
38
39      for (auto [u,d] : adj[v]) {
40        if (!visited[u]) {
41          visited[u] = true;
42          up[u][0] = v;
43          q.push(mp(u, depth + 1));
44        }
45      }
46    }
47 }
48
49 void find_level_peso() {
50    queue<pii> q;
51
52    q.push(mp(1, 0));
53    visited[1] = true;
54
55    while (!q.empty()) {
56      auto [v, depth] = q.front();
57      q.pop();
58      level_peso[v] = depth;
59
60      for (auto [u,d] : adj[v]) {
61        if (!visited[u]) {
62          visited[u] = true;
63          up[u][0] = v;
64          q.push(mp(u, depth + d));
65        }
66      }
67    }
68 }
69
70 int lca(int a, int b) {
71    // get the nodes to the same level
72    int mn = min(level[a], level[b]);
73
74    for (int j = 0; j <= BITS; j++) {
75      if (a != -1 && ((level[a] - mn) & (1 << j))) a
         = up[a][j];
76      if (b != -1 && ((level[b] - mn) & (1 << j))) b
         = up[b][j];
77    }
78
79    // special case
80    if (a == b) return a;
81
82    // binary search
83    for (int j = BITS; j >= 0; j--) {
84      if (up[a][j] != up[b][j]) {
85        a = up[a][j];
86        b = up[b][j];
87      }
88    }
89    return up[a][0];
90 }
91
92 void preprocess() {
93    visited = vector<bool>(MAX, false);
94    find_level();
95    visited = vector<bool>(MAX, false);
96    find_level_peso();
97
98    for (int j = 1; j <= BITS; j++) {
99      for (int i = 1; i <= n; i++) {
100       if (up[i][j - 1] != -1) up[i][j] = up[up[i][j -
         1]][j - 1];
101     }
```

```
102   }
103 }
```

6.9   Kuhn

```
1  // Description
2  // Matching algorithm for unweighted bipartite graph
        ::
3
4  // Problem:
5  // https://codeforces.com/gym/104252/problem/H
6
7  // Complexity:
8  // O(V * E) in which V is the number of vertexes and
        E is the number of edges
9
10 // Notes:
11 // Indexed at zero
12
13 int n, k;
14 // adjacency list
15 vector<vector<int>> g;
16 vector<int> mt;
17 vector<bool> used;
18
19 bool try_kuhn(int v) {
20     if (used[v])
21         return false;
22     used[v] = true;
23     for (int to : g[v]) {
24         if (mt[to] == -1 || try_kuhn(mt[to])) {
25             mt[to] = v;
26             return true;
27         }
28     }
29     return false;
30 }
31
32 int main() {
33     // ... reading the graph g ...
34
35     mt.assign(k, -1);
36     vector<bool> used1(n, false);
37     for (int v = 0; v < n; ++v) {
38         for (int to : g[v]) {
39             if (mt[to] == -1) {
40                 mt[to] = v;
41                 used1[v] = true;
42                 break;
43             }
44         }
45     }
46     for (int v = 0; v < n; ++v) {
47         if (used1[v])
48             continue;
49         used.assign(n, false);
50         try_kuhn(v);
51     }
52
53     for (int i = 0; i < k; ++i)
54         if (mt[i] != -1)
55             printf("%d %d\n", mt[i] + 1, i + 1);
56 }
```

6.10   Eulerian Directed

```
1  // Description:
2  // Hierholzer's Algorithm
3  // An Eulerian path is a path that passes through
        every edge exactly once.
4  // An Eulerian circuit is an Eulerian path that
        starts and ends on the same node.
```

```cpp
5
6  // An Eulerian path exists in an directed graph if
       the indegree and outdegree is equal
7  // for every node (not counting self-edges)
8  // except for possibly exactly one node that have
       outdegree - indegree = 1
9  // and one node that has indegree - outdegreee = 1 (
       start and end nodes).
10 // An Eulerian circuit exists in an directed graph if
       the indegree and outdegree is equal for every
       node.
11
12 // The graph has to be conected (except for isolated
       nodes which are allowed because there
13 // are no edges connected to them).
14
15 // Problem:
16 // https://cses.fi/problemset/task/1693
17
18 // Complexity:
19 // O(E) where E is the number of edges
20
21 // How to use
22 // Check whether the path exists before trying to
       find it
23 // Find the root - any node that has at least 1
       outgoing edge
24 // (if the problem requires that you start from a
       node v, the root will be the node v)
25 // Count the degree;
26 //
27 // for (int i = 0; i < m; i++) {
28 //   int a, b; cin >> a >> b;
29 //   adj[a].pb(b);
30 //   root = a;
31 //   outdegree[a]++; indegree[b]++;
32 // }
33
34 // Notes
35 // It works when there are self loops, but not when
       there are multiple edges
36
37 vector<bool> visited;
38 vector<int> outdegree, indegree;
39 vector<vector<int>> adj, undir;
40
41 void dfs(int v) {
42   visited[v] = true;
43   for (auto u : undir[v]) {
44     if (!visited[u]) dfs(u);
45   }
46 }
47
48 int is_eulerian(int n, int root, int &start, int& end
       ) {
49   start = -1, end = -1;
50   if (n == 1) return 2; // only one node
51   visited.assign(n + 1, false);
52   dfs(root);
53
54   for (int i = 1; i <= n; i++) {
55     if (!visited[i] && (i == n || i == 1 || outdegree
       [i] + indegree[i] > 0)) return 0;
56   }
57
58   // start => node with indegree - outdegree = 1
59   // end => node with outdegree - indegree = 1
60   for (int i = 1; i <= n; i++) {
61     if (start == -1 && indegree[i] - outdegree[i] ==
       1) start = i;
62     else if (end == -1 && outdegree[i] - indegree[i]
       == 1) end = i;
63     else if (indegree[i] != outdegree[i]) return 0;
64   }
65
66   if (start == -1 && end == -1) {start = root; end =
       root; return 2;} // has eulerian circuit and path
67   if (start != -1 && end != -1) {swap(start, end);
       return 1;} // has eulerian path
68   return 0; // no eulerian path nor circuit
69 }
70
71 vector<int> path;
72
73 void dfs_path(int v) {
74   visited[v] = true;
75
76   while (outdegree[v] != 0) {
77     int u = adj[v][--outdegree[v]];
78     int next_edge = adj[v][outdegree[v]];
79     dfs_path(next_edge);
80   }
81   path.pb(v);
82 }
83
84 void find_path(int n, int start) {
85   path.clear();
86   visited.assign(n + 1, false);
87   dfs_path(start);
88   reverse(path.begin(), path.end());
89 }
```

6.11   Bellman Ford

```cpp
1  // Description:
2  // Finds the shortest path from a vertex v to any
       other vertex
3
4  // Problem:
5  // https://cses.fi/problemset/task/1673
6
7  // Complexity:
8  // O(n * m)
9
10 struct Edge {
11   int a, b, cost;
12   Edge(int a, int b, int cost) : a(a), b(b), cost(
       cost) {}
13 };
14
15 int n, m;
16 vector<Edge> edges;
17 const int INF = 1e9+10;
18
19 void bellman_ford(int v, int t) {
20   vector<int> d(n + 1, INF);
21   d[v] = 0;
22   vector<int> p(n + 1, -1);
23
24   for (;;) {
25     bool any = false;
26     for (Edge e : edges) {
27       if (d[e.a] >= INF) continue;
28       if (d[e.b] > d[e.a] + e.cost) {
29         d[e.b] = d[e.a] + e.cost;
30         p[e.b] = e.a;
31         any = true;
32       }
33     }
34     if (!any) break;
35   }
36
37   if (d[t] == INF)
38     cout << "No path from " << v << " to " << t << ".
       ";
39   else {
```

```cpp
    vector<int> path;
    for (int cur = t; cur != -1; cur = p[cur]) {
      path.push_back(cur);
    }
    reverse(path.begin(), path.end());

    cout << "Path from " << v << " to " << t << ": ";
    for (int u : path) {
      cout << u << ' ';
    }
  }
}
```

6.12 Dinic

```cpp
// Description:
// 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

// Problem:
// https://codeforces.com/gym/103708/problem/J

// Complexity:
// O(V^2 * E) where V is the number of vertex and E
    is the number of edges

// Unit network
// A unit network is a network in which for any
    vertex except source and sink either incoming or
    outgoing edge is unique and has unit capacity (
    matching problem).
// Complexity on unit networks: O(E * sqrt(V))

// Unity capacity networks
// A more generic settings when all edges have unit
    capacities, but the number of incoming and
    outgoing edges is unbounded
// Complexity on unity capacity networks: O(E * sqrt(
    E))

// How to use:
// Dinic dinic = Dinic(num_vertex, source, sink);
// dinic.add_edge(vertex1, vertex2, capacity);
// cout << dinic.max_flow() << '\n';

#include <bits/stdc++.h>

#define pb push_back
#define mp make_pair
#define pii pair<int, int>
#define ff first
#define ss second
#define ll long long

using namespace std;

const ll INF = 1e18+10;

struct Edge {
    int from;
    int to;
    ll capacity;
    ll flow;
    Edge* residual;

    Edge() {}

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

    ll get_capacity() {
        return capacity - flow;
    }

    ll get_flow() {
        return flow;
    }

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

    void reverse(ll bottleneck) {
        flow -= bottleneck;
        residual->flow += bottleneck;
    }

    bool operator<(const Edge& e) const {
        return true;
    }
};

struct Dinic {
    int source;
    int sink;
    int nodes;
    ll flow;
    vector<vector<Edge*>> adj;
    vector<int> level;
    vector<int> next;
    vector<int> reach;
    vector<bool> visited;
    vector<vector<int>> path;

    Dinic(int source, int sink, int nodes) : source(
    source), sink(sink), nodes(nodes) {
        adj.resize(nodes + 1);
    }

    void add_edge(int from, int to, ll capacity) {
        Edge* e1 = new Edge(from, to, capacity);
        Edge* e2 = new Edge(to, from, 0);
        // Edge* e2 = new Edge(to, from, capacity);
        e1->residual = e2;
        e2->residual = e1;
        adj[from].pb(e1);
        adj[to].pb(e2);
    }

    bool bfs() {
        level.assign(nodes + 1, -1);
        queue<int> q;
        q.push(source);
        level[source] = 0;

        while (!q.empty()) {
            int node = q.front();
            q.pop();

            for (auto e : adj[node]) {
                if (level[e->to] == -1 && e->
    get_capacity() > 0) {
                    level[e->to] = level[e->from] +
    1;
                    q.push(e->to);
                }
            }
        }

        return level[sink] != -1;
    }
```

```cpp
ll dfs(int v, ll flow) {
    if (v == sink)
        return flow;

    int sz = adj[v].size();
    for (int i = next[v]; i < sz; i++) {
        Edge* e = adj[v][i];
        if (level[e->to] == level[e->from] + 1 &&
 e->get_capacity() > 0) {
            ll bottleneck = dfs(e->to, min(flow,
 e->get_capacity()));
            if (bottleneck > 0) {
                e->augment(bottleneck);
                return bottleneck;
            }
        }

        next[v] = i + 1;
    }

    return 0;
}

ll max_flow() {
    flow = 0;
    while(bfs()) {
        next.assign(nodes + 1, 0);
        ll sent = -1;
        while (sent != 0) {
            sent = dfs(source, INF);
            flow += sent;
        }
    }
    return flow;
}

void reachable(int v) {
    visited[v] = true;

    for (auto e : adj[v]) {
        if (!visited[e->to] && e->get_capacity()
 > 0) {
            reach.pb(e->to);
            visited[e->to] = true;
            reachable(e->to);
        }
    }
}

void print_min_cut() {
    reach.clear();
    visited.assign(nodes + 1, false);
    reach.pb(source);
    reachable(source);

    for (auto v : reach) {
        for (auto e : adj[v]) {
            if (!visited[e->to] && e->
get_capacity() == 0) {
                cout << e->from << ' ' << e->to
 << '\n';
            }
        }
    }
}

ll build_path(int v, int id, ll flow) {
    visited[v] = true;
    if (v == sink) {
        return flow;
    }

    for (auto e : adj[v]) {
        if (!visited[e->to] && e->get_flow() > 0)
 {
            visited[e->to] = true;
            ll bottleneck = build_path(e->to, id,
 min(flow, e->get_flow()));
            if (bottleneck > 0) {
                path[id].pb(e->to);
                e->reverse(bottleneck);
                return bottleneck;
            }
        }
    }

    return 0;
}

void print_flow_path() {
    path.clear();
    ll sent = -1;
    int id = -1;
    while (sent != 0) {
        visited.assign(nodes + 1, false);
        path.pb(vector<int>{});
        sent = build_path(source, ++id, INF);
        path[id].pb(source);
    }
    path.pop_back();

    for (int i = 0; i < id; i++) {
        cout << path[i].size() << '\n';
        reverse(path[i].begin(), path[i].end());
        for (auto e : path[i]) {
            cout << e << ' ';
        }
        cout << '\n';
    }
}
};

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

    int n, m; cin >> n >> m;

    Dinic dinic = Dinic(1, n, n);

    for (int i = 1; i <= m; i++) {
        int v, u; cin >> v >> u;
        dinic.add_edge(v, u, 1);
    }

    cout << dinic.max_flow() << '\n';
    // dinic.print_min_cut();
    // dinic.print_flow_path();

    return 0;
}
```

6.13 2sat

```cpp
// Description:
// Solves expression of the type (a v b) ^ (c v d) ^
    (e v f)

// Problem:
// https://cses.fi/problemset/task/1684

// Complexity:
// O(n + m) where n is the number of variables and m
    is the number of clauses
```

```cpp
10  #include <bits/stdc++.h>
11  #define pb push_back
12  #define mp make_pair
13  #define pii pair<int, int>
14  #define ff first
15  #define ss second
16
17  using namespace std;
18
19  struct SAT {
20      int nodes;
21      int curr = 0;
22      int component = 0;
23      vector<vector<int>> adj;
24      vector<vector<int>> rev;
25      vector<vector<int>> condensed;
26      vector<pii> departure;
27      vector<bool> visited;
28      vector<int> scc;
29      vector<int> order;
30
31      // 1 to nodes
32      // nodes + 1 to 2 * nodes
33      SAT(int nodes) : nodes(nodes) {
34          adj.resize(2 * nodes + 1);
35          rev.resize(2 * nodes + 1);
36          visited.resize(2 * nodes + 1);
37          scc.resize(2 * nodes + 1);
38      }
39
40      void add_imp(int a, int b) {
41          adj[a].pb(b);
42          rev[b].pb(a);
43      }
44
45      int get_not(int a) {
46          if (a > nodes) return a - nodes;
47          return a + nodes;
48      }
49
50      void add_or(int a, int b) {
51          add_imp(get_not(a), b);
52          add_imp(get_not(b), a);
53      }
54
55      void add_nor(int a, int b) {
56          add_or(get_not(a), get_not(b));
57      }
58
59      void add_and(int a, int b) {
60          add_or(get_not(a), b);
61          add_or(a, get_not(b));
62          add_or(a, b);
63      }
64
65      void add_nand(int a, int b) {
66          add_or(get_not(a), b);
67          add_or(a, get_not(b));
68          add_or(get_not(a), get_not(b));
69      }
70
71      void add_xor(int a, int b) {
72          add_or(a, b);
73          add_or(get_not(a), get_not(b));
74      }
75
76      void add_xnor(int a, int b) {
77          add_or(get_not(a), b);
78          add_or(a, get_not(b));
79      }
80
81      void departure_time(int v) {
82          visited[v] = true;
83
84          for (auto u : adj[v]) {
85              if (!visited[u]) departure_time(u);
86          }
87
88          departure.pb(mp(++curr, v));
89      }
90
91      void find_component(int v, int component) {
92          scc[v] = component;
93          visited[v] = true;
94
95          for (auto u : rev[v]) {
96              if (!visited[u]) find_component(u,
    component);
97          }
98      }
99
100     void topological_order(int v) {
101         visited[v] = true;
102
103         for (auto u : condensed[v]) {
104             if (!visited[u]) topological_order(u);
105         }
106
107         order.pb(v);
108     }
109
110     bool is_possible() {
111         component = 0;
112         for (int i = 1; i <= 2 * nodes; i++) {
113             if (!visited[i]) departure_time(i);
114         }
115
116         sort(departure.begin(), departure.end(),
    greater<pii>());
117
118         visited.assign(2 * nodes + 1, false);
119
120         for (auto [_, node] : departure) {
121             if (!visited[node]) find_component(node,
    ++component);
122         }
123
124         for (int i = 1; i <= nodes; i++) {
125             if (scc[i] == scc[i + nodes]) return
    false;
126         }
127
128         return true;
129     }
130
131     int find_value(int e, vector<int> &ans) {
132         if (e > nodes && ans[e - nodes] != 2) return
    !ans[e - nodes];
133         if (e <= nodes && ans[e + nodes] != 2) return
     !ans[e + nodes];
134         return 0;
135     }
136
137     vector<int> find_ans() {
138         condensed.resize(component + 1);
139
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 }
```

6.14   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 }
```

6.15   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 }
```

6.16 Blossom

```
1  // Description:
2  // Matching algorithm for general graphs (non-
       bipartite)
3
4  // Problem:
5  // https://acm.timus.ru/problem.aspx?space=1&num=1099
6
7  // Complexity:
8  // O (n ^3)
9
10 // vector<pii> Blossom(vector<vector<int>>& graph) {
11 vector<int> Blossom(vector<vector<int>>& graph) {
12   int n = graph.size(), timer = -1;
13   vector<int> mate(n, -1), label(n), parent(n),
14              orig(n), aux(n, -1), q;
15   auto lca = [&](int x, int y) {
16     for (timer++; ; swap(x, y)) {
17       if (x == -1) continue;
18       if (aux[x] == timer) return x;
19       aux[x] = timer;
20       x = (mate[x] == -1 ? -1 : orig[parent[mate[x
     ]]]);
21     }
22   };
23   auto blossom = [&](int v, int w, int a) {
24     while (orig[v] != a) {
25       parent[v] = w; w = mate[v];
26       if (label[w] == 1) label[w] = 0, q.push_back(w)
     ;
27       orig[v] = orig[w] = a; v = parent[w];
28     }
29   };
30   auto augment = [&](int v) {
31     while (v != -1) {
32       int pv = parent[v], nv = mate[pv];
33       mate[v] = pv; mate[pv] = v; v = nv;
34     }
35   };
36   auto bfs = [&](int root) {
37     fill(label.begin(), label.end(), -1);
38     iota(orig.begin(), orig.end(), 0);
39     q.clear();
40     label[root] = 0; q.push_back(root);
41     for (int i = 0; i < (int)q.size(); ++i) {
42       int v = q[i];
43       for (auto x : graph[v]) {
44         if (label[x] == -1) {
45           label[x] = 1; parent[x] = v;
46           if (mate[x] == -1)
47             return augment(x), 1;
48           label[mate[x]] = 0; q.push_back(mate[x]);
49         } else if (label[x] == 0 && orig[v] != orig[x
     ]) {
50           int a = lca(orig[v], orig[x]);
51           blossom(x, v, a); blossom(v, x, a);
52         }
53       }
54     }
55     return 0;
56   };
57   // Time halves if you start with (any) maximal
       matching.
58   for (int i = 0; i < n; i++)
59     if (mate[i] == -1)
60       bfs(i);
61   return mate;
```

```
62
63   /*
64   vector<bool> used(n, false);
65   vector<pii> ans;
66   for (int i = 0; i < n; i++) {
67     if (matching[i] == -1 || used[i]) continue;
68     used[i] = true;
69     used[matching[i]] = true;
70     ans.emplace_back(i, matching[i]);
71   }
72   return ans;
73   */
74 }
```

6.17 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(){
```

22

```cpp
        ios::sync_with_stdio(false);
        cin.tie(NULL);

        cin >> n;
        adj.resize(n + 1);
        ans.resize(n + 1);
        removed.resize(n + 1);
        subtree_size.resize(n + 1);

        for (int i = 1; i <= n - 1; i++) {
            int u, v; cin >> u >> v;
            adj[u].insert(v);
            adj[v].insert(u);
        }

        solve(1, 'A');

        if (!flag) cout << "Impossible!\n";
        else {
            for (int i = 1; i <= n; i++) {
                cout << ans[i] << ' ';
            }
            cout << '\n';
        }

        return 0;
}
```

6.18   Tarjan Bridge

```cpp
// Description:
// Find a bridge in a connected unidirected graph
// A bridge is an edge so that if you remove that
//    edge the graph is no longer connected

// Problem:
// https://cses.fi/problemset/task/2177/

// Complexity:
// O(V + E) where V is the number of vertices and E
//    is the number of edges

int n;
vector<vector<int>> adj;

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v]) {
                IS_BRIDGE(v, to);
            }
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i, -1);
    }
}
```

6.19   Hld Vertex

```cpp
// Description:
// Make queries and updates between two vertexes on a
//     tree
// Query path - query path (a, b) inclusive
// Update path - update path (a, b) inclusive
// Query subtree - query subtree of a
// Update subtree - update subtree of a
// Update - update vertex or edge
// Lca - get lowest common ancestor of a and b
// Search - perform a binary search to find the last
//     node with a certain property
// on the path from a to the root

// Problem:
// https://codeforces.com/gym/101908/problem/L

// Complexity:
// O(log ^2 n) for both query and update

// How to use:
// HLD hld = HLD(n + 1, adj)

// Notes
// Change the root of the tree on the constructor if
//     it's different from 1
// Use together with Segtree

typedef long long ftype;

struct HLD {
    vector<int> parent;
    vector<int> pos;
    vector<int> head;
    vector<int> subtree_size;
    vector<int> level;
    vector<int> heavy_child;
    vector<ftype> subtree_weight;
    vector<ftype> path_weight;
    vector<vector<int>> adj;
    vector<int> at;
    Segtree seg = Segtree(0);
    int cpos;
    int n;
    int root;
    vector<vector<int>> up;

    HLD() {}

    HLD(int n, vector<vector<int>>& adj, int root = 1)
        : adj(adj), n(n), root(root) {
        seg = Segtree(n);
        cpos = 0;
        at.resize(n);
        parent.resize(n);
        pos.resize(n);
        head.resize(n);
        subtree_size.assign(n, 1);
        level.assign(n, 0);
        heavy_child.assign(n, -1);
        parent[root] = -1;
        dfs(root, -1);
        decompose(root, -1);
    }

    void dfs(int v, int p) {
        parent[v] = p;
        if (p != -1) level[v] = level[p] + 1;
```

```cpp
  64      for (auto u : adj[v]) {
  65        if (u != p) {
  66          dfs(u, v);
  67          subtree_size[v] += subtree_size[u];
  68          if (heavy_child[v] == -1 || subtree_size[u] >
          subtree_size[heavy_child[v]]) heavy_child[v] = u
          ;
  69        }
  70      }
  71    }
  72
  73    void decompose(int v, int chead) {
  74      // start a new path
  75      if (chead == -1) chead = v;
  76
  77      // consecutive ids in the hld path
  78      at[cpos] = v;
  79      pos[v] = cpos++;
  80      head[v] = chead;
  81
  82      // if not a leaf
  83      if (heavy_child[v] != -1) decompose(heavy_child[v
        ], chead);
  84
  85      // light child
  86      for (auto u : adj[v]){
  87        // start new path
  88        if (u != parent[v] && u != heavy_child[v])
        decompose(u, -1);
  89      }
  90    }
  91
  92    ftype query_path(int a, int b) {
  93      if(pos[a] < pos[b]) swap(a, b);
  94
  95      if(head[a] == head[b]) return seg.query(pos[b],
        pos[a]);
  96      return seg.f(seg.query(pos[head[a]], pos[a]),
        query_path(parent[head[a]], b));
  97    }
  98
  99    // iterative
 100    /*ftype query_path(int a, int b) {
 101      ftype ans = 0;
 102
 103      while (head[a] != head[b]) {
 104        if (level[head[a]] > level[head[b]]) swap(a, b)
        ;
 105        ans = seg.merge(ans, seg.query(pos[head[b]],
        pos[b]));
 106        b = parent[head[b]];
 107      }
 108
 109      if (level[a] > level[b]) swap(a, b);
 110      ans = seg.merge(ans, seg.query(pos[a], pos[b]));
 111      return ans;
 112    }*/
 113
 114    ftype query_subtree(int a) {
 115      return seg.query(pos[a], pos[a] + subtree_size[a]
        - 1);
 116    }
 117
 118    void update_path(int a, int b, int x) {
 119      if(pos[a] < pos[b]) swap(a, b);
 120
 121      if(head[a] == head[b]) return (void)seg.update(
        pos[b], pos[a], x);
 122      seg.update(pos[head[a]], pos[a], x); update_path(
        parent[head[a]], b, x);
 123    }
 124
 125    void update_subtree(int a, int val) {
```

```cpp
 126      seg.update(pos[a], pos[a] + subtree_size[a] - 1,
        val);
 127    }
 128
 129    void update(int a, int val) {
 130      seg.update(pos[a], pos[a], val);
 131    }
 132
 133    //edge
 134    void update(int a, int b, int val) {
 135      if (level[a] > level[b]) swap(a, b);
 136      update(b, val);
 137    }
 138
 139    int lca(int a, int b) {
 140      if(pos[a] < pos[b]) swap(a, b);
 141      return head[a] == head[b] ? b : lca(parent[head[a
        ]], b);
 142    }
 143
 144    void search(int a) {
 145      a = parent[a];
 146      if (a == -1) return;
 147      if (seg.query(pos[head[a]], pos[head[a]]+
        subtree_size[head[a]]-1) + pos[a]-pos[head[a]]+1
        == subtree_size[head[a]]) {
 148        seg.update(pos[head[a]], pos[a], 1);
 149        return search(parent[head[a]]);
 150      }
 151      int l = pos[head[a]], r = pos[a]+1;
 152      while (l < r) {
 153        int m = (l+r)/2;
 154        if (seg.query(m, m+subtree_size[at[m]]-1) + pos
        [a]-m+1 == subtree_size[at[m]]) {
 155          r = m;
 156        }
 157        else l = m+1;
 158      }
 159      seg.update(l, pos[a], 1);
 160    }
 161
 162    /* k-th ancestor of x
 163    int x, k; cin >> x >> k;
 164
 165    for (int b = 0; b <= BITS; b++) {
 166      if (x != -1 && (k & (1 << b))) {
 167        x = up[x][b];
 168      }
 169    }
 170
 171    cout << x << '\n';
 172    */
 173    void preprocess() {
 174      up.assign(n + 1, vector<int>(31, -1));
 175
 176      for (int i = 1; i < n; i++) {
 177        up[i][0] = parent[i];
 178      }
 179
 180      for (int i = 1; i < n; i++) {
 181        for (int j = 1; j <= 30; j++) {
 182          if (up[i][j - 1] != -1) up[i][j] = up[up[i][j
        - 1]][j - 1];
 183        }
 184      }
 185    }
 186
 187    int getKth(int p, int q, int k){
 188      int a = lca(p,q), d;
 189
 190      if( a == p ){
 191        d = level[q] - level[p] + 1;
 192        swap(p,q);
```

```
193        k = d - k + 1 ;
194    }
195    else if( a == q ) ;
196    else {
197        if( k > level[p] - level[a] + 1 ) {
198            d = level[p] + level[q] - 2 * level[a] +
       1 ;
199            k = d - k + 1 ;
200            swap(p,q);
201        }
202        else ;
203    }
204    int lg ; for( lg = 1 ; (1 << lg) <= level[p] ; ++
       lg ); lg--;
205    k--;
206    for( int i = lg ; i >= 0 ; i-- ){
207        if( (1 << i) <= k ){
208            p = up[p][i];
209            k -= ( 1 << i );
210        }
211    }
212    return p ;
213 }
214 };
```

6.20   Small To Large

```
1  // Problem:
2  // https://codeforces.com/contest/600/problem/E
3
4  void process_colors(int curr, int parent) {
5
6    for (int n : adj[curr]) {
7      if (n != parent) {
8        process_colors(n, curr);
9
10            if (colors[curr].size() < colors[n].size
     ()) {
11                sum_num[curr] = sum_num[n];
12                vmax[curr] = vmax[n];
13          swap(colors[curr], colors[n]);
14        }
15
16      for (auto [item,vzs] : colors[n]) {
17              if(colors[curr][item]+vzs > vmax[curr
     ]){
18                  vmax[curr] = colors[curr][item] +
      vzs;
19                  sum_num[curr] = item;
20              }
21              else if(colors[curr][item]+vzs ==
     vmax[curr]){
22                  sum_num[curr] += item;
23              }
24
25              colors[curr][item] += vzs;
26      }
27    }
28  }
29
30 }
31
32
33 int32_t main() {
34
35  int n; cin >> n;
36
37  for (int i = 1; i <= n; i++) {
38    int a; cin >> a;
39    colors[i][a] = 1;
40      vmax[i] = 1;
41      sum_num[i] = a;
42  }
```

```
43
44  for (int i = 1; i < n; i++) {
45    int a, b; cin >> a >> b;
46
47    adj[a].push_back(b);
48    adj[b].push_back(a);
49  }
50
51  process_colors(1, 0);
52
53  for (int i = 1; i <= n; i++) {
54    cout << sum_num[i] << (i < n ? " " : "\n");
55  }
56
57    return 0;
58
59 }
60
```

6.21   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 }
```

6.22   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      int n = adj.size();
7      d.assign(n, INF);
8      p.assign(n, -1);
9
10     d[s] = 0;
11     set<pair<int, int>> q;
12     q.insert({0, s});
13     while (!q.empty()) {
14         int v = q.begin()->second;
15         q.erase(q.begin());
16
```

```
17          for (auto edge : adj[v]) {
18              int to = edge.first;
19              int len = edge.second;
20
21              if (d[v] + len < d[to]) {
22                  q.erase({d[to], to});
23                  d[to] = d[v] + len;
24                  p[to] = v;
25                  q.insert({d[to], to});
26              }
27          }
28      }
29  }
30
31  vector<int> restore_path(int s, int t) {
32      vector<int> path;
33
34      for (int v = t; v != s; v = p[v])
35          path.push_back(v);
36      path.push_back(s);
37
38      reverse(path.begin(), path.end());
39      return path;
40  }
41
42  int adj[MAX][MAX];
43  int dist[MAX];
44  int minDistance(int dist[], bool sptSet[], int V) {
45      int min = INT_MAX, min_index;
46
47      for (int v = 0; v < V; v++)
48          if (sptSet[v] == false && dist[v] <= min)
49              min = dist[v], min_index = v;
50
51      return min_index;
52  }
53
54  void dijkstra(int src, int V) {
55
56      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
68          for (int v = 0; v < V; v++)
69              if (!sptSet[v] && adj[u][v]
70                  && dist[u] != INT_MAX
71                  && dist[u] + adj[u][v] < dist[v])
72                  dist[v] = dist[u] + adj[u][v];
73      }
74  }
```

6.23 Kruskall

```
1  struct DSU {
2      int n;
3      vector<int> link, sizes;
4
5      DSU(int n) {
6          this->n = n;
7          link.assign(n+1, 0);
8          sizes.assign(n+1, 1);
9
10         for (int i = 0; i <= n; i++)
11             link[i] = i;
12     }
```

```
13
14      int find(int x) {
15          while (x != link[x])
16              x = link[x];
17
18          return x;
19      }
20
21      bool same(int a, int b) {
22          return find(a) == find(b);
23      }
24
25      void unite(int a, int b) {
26          a = find(a);
27          b = find(b);
28
29          if (a == b) return;
30
31          if (sizes[a] < sizes[b])
32              swap(a, b);
33
34          sizes[a] += sizes[b];
35          link[b] = a;
36      }
37  };
38
39  struct Edge {
40      int u, v;
41      long long weight;
42
43      Edge() {}
44
45      Edge(int u, int v, long long weight) : u(u), v(v)
       , weight(weight) {}
46
47      bool operator<(const Edge& other) const {
48          return weight < other.weight;
49      }
50
51      bool operator>(const Edge& other) const {
52          return weight > other.weight;
53      }
54  };
55
56  vector<Edge> kruskal(vector<Edge> edges, int n) {
57      vector<Edge> result; // arestas da MST
58      long long cost = 0;
59
60      sort(edges.begin(), edges.end());
61
62      DSU dsu(n);
63
64      for (auto e : edges) {
65          if (!dsu.same(e.u, e.v)) {
66              cost += e.weight;
67              result.push_back(e);
68              dsu.unite(e.u, e.v);
69          }
70      }
71
72      return result;
73  }
```

6.24 Hungarian

```
1  // Description:
2  // A matching algorithm for weighted bipartite graphs
       that returns
3  // a perfect match
4
5  // Problem:
6  // https://codeforces.com/gym/103640/problem/H
7
```

```
8  // Complexity:
9  // O(V ^ 3) in which V is the number of vertexs
10
11 // Notes:
12 // Indexed at 1
13
14 // n is the number of items on the right side and m
       the number of items
15 // on the left side of the graph
16
17 // Returns minimum assignment cost and which items
       were matched
18
19 pair<int, vector<pii>> hungarian(int n, int m, vector
       <vector<int>> A) {
20   vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
21   for (int i=1; i<=n; ++i) {
22     p[0] = i;
23     int j0 = 0;
24     vector<int> minv (m+1, INF);
25     vector<char> used (m+1, false);
26     do {
27       used[j0] = true;
28       int i0 = p[j0], delta = INF, j1;
29       for (int j=1; j<=m; ++j)
30         if (!used[j]) {
31           int cur = A[i0][j]-u[i0]-v[j];
32           if (cur < minv[j])
33             minv[j] = cur, way[j] = j0;
34           if (minv[j] < delta)
35             delta = minv[j], j1 = j;
36         }
37       for (int j=0; j<=m; ++j)
38         if (used[j])
39           u[p[j]] += delta, v[j] -= delta;
40         else
41           minv[j] -= delta;
42       j0 = j1;
43     } while (p[j0] != 0);
44     do {
45       int j1 = way[j0];
46       p[j0] = p[j1];
47       j0 = j1;
48     } while (j0);
49   }
50
51   vector<pair<int, int>> result;
52   for (int i = 1; i <= m; ++i){
53     result.push_back(make_pair(p[i], i));
54   }
55
56   int C = -v[0];
57
58   return mp(C, result);
59 }
```

6.25 Negative Cycle

```
1  // Description
2  // Detects any cycle in which the sum of edge weights
       is negative.
3  // Alternatively, we can detect whether there is a
       negative cycle
4  // starting from a specific vertex.
5
6  // Problem:
7  // https://cses.fi/problemset/task/1197
8
9  // Complexity:
10 // O(n * m)
11
12 // Notes
```

```
13 // In order to consider only the negative cycles
       located on the path from a to b,
14 // Reverse the graph, run a dfs from node b and mark
       the visited nodes
15 // Consider only the edges that connect to visited
       nodes when running bellman-ford
16 // on the normal graph
17
18 struct Edge {
19   int a, b, cost;
20   Edge(int a, int b, int cost) : a(a), b(b), cost(
       cost) {}
21 };
22
23 int n, m;
24 vector<Edge> edges;
25 const int INF = 1e9+10;
26
27 void negative_cycle() {
28   // uncomment to find negative cycle starting from a
       vertex v
29   // vector<int> d(n + 1, INF);
30   // d[v] = 0;
31   vector<int> d(n + 1, 0);
32   vector<int> p(n + 1, -1);
33   int x;
34   // uncomment to find all negative cycles
35   // // set<int> s;
36   for (int i = 1; i <= n; ++i) {
37     x = -1;
38     for (Edge e : edges) {
39       // if (d[e.a] >= INF) continue;
40       if (d[e.b] > d[e.a] + e.cost) {
41         // d[e.b] = max(-INF, d[e.a] + e.cost);
42         d[e.b] = d[e.a] + e.cost;
43         p[e.b] = e.a;
44         x = e.b;
45         // // s.insert(e.b);
46       }
47     }
48   }
49
50   if (x == -1)
51     cout << "NO\n";
52   else {
53     // // int y = all nodes in set s
54     int y = x;
55     for (int i = 1; i <= n; ++i) {
56       y = p[y];
57     }
58
59     vector<int> path;
60     for (int cur = y;; cur = p[cur]) {
61       path.push_back(cur);
62       if (cur == y && path.size() > 1) break;
63     }
64     reverse(path.begin(), path.end());
65
66     cout << "YES\n";
67     for (int u : path)
68       cout << u << ' ';
69     cout << '\n';
70   }
71 }
```

7 Geometry

7.1 Shoelace Boundary

```
1 // Description
2 // Shoelace formula finds the area of a polygon
```

```
3  // Boundary points return the number of integer
       points on the edges of a polygon
4  // not counting the vertexes
5
6  // Problem
7  // https://codeforces.com/gym/101873/problem/G
8
9  // Complexity
10 // O(n)
11
12 // before dividing by two
13 int shoelace(vector<point> & points) {
14     int n = points.size();
15     vector<point> v(n + 2);
16
17     for (int i = 1; i <= n; i++) {
18         v[i] = points[i - 1];
19     }
20     v[n + 1] = points[0];
21
22     int sum = 0;
23     for (int i = 1; i <= n; i++) {
24         sum += (v[i].x * v[i + 1].y - v[i + 1].x * v[
       i].y);
25     }
26
27     sum = abs(sum);
28     return sum;
29 }
30
31 int boundary_points(vector<point> & points) {
32     int n = points.size();
33     vector<point> v(n + 2);
34
35     for (int i = 1; i <= n; i++) {
36         v[i] = points[i - 1];
37     }
38     v[n + 1] = points[0];
39
40     int ans = 0;
41     for (int i = 1; i <= n; i++) {
42         if (v[i].x == v[i + 1].x) ans += abs(v[i].y -
        v[i + 1].y) - 1;
43         else if (v[i].y == v[i + 1].y) ans += abs(v[i
       ].x - v[i + 1].x) - 1;
44         else ans += gcd(abs(v[i].x - v[i + 1].x), abs
       (v[i].y - v[i + 1].y)) - 1;
45     }
46     return points.size() + ans;
47 }
```

7.2 Inside Polygon

```
1  // Description
2  // Checks if a given point is inside, outside or on
       the boundary of a polygon
3
4  // Problem
5  // https://cses.fi/problemset/task/2192/
6
7  // Complexity
8  // O(n)
9
10 int inside(vp &p, point pp){
11     // 1 - inside / 0 - boundary / -1 - outside
12     int n = p.size();
13     for(int i=0;i<n;i++){
14         int j = (i+1)%n;
15         if(line({p[i], p[j]}).inside_seg(pp))
16             return 0; // boundary
17     }
18     int inter = 0;
19     for(int i=0;i<n;i++){
```

```
20         int j = (i+1)%n;
21         if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
       [i], p[j], pp)==1)
22             inter++; // up
23         else if(p[j].x <= pp.x and pp.x < p[i].x and
       ccw(p[i], p[j], pp)==-1)
24             inter++; // down
25     }
26
27     if(inter%2==0) return -1; // outside
28     else return 1; // inside
29 }
```

7.3 Closest Pair Points

```
1  // Description
2  // Find the squared distance between the closest two
       points among n points
3  // Also finds which pair of points is closest (could
       be more than one)
4
5  // Problem
6  // https://cses.fi/problemset/task/2194/
7
8  // Complexity
9  // O(n log n)
10
11 ll closest_pair_points(vp &vet){
12     pair<point, point> ans;
13     int n = vet.size();
14     sort(vet.begin(), vet.end());
15     set<point> s;
16
17     ll best_dist = LLONG_MAX;
18     int j=0;
19     for(int i=0;i<n;i++){
20         ll d = ceil(sqrt(best_dist));
21         while(j<n and vet[i].x-vet[j].x >= d){
22             s.erase(point(vet[j].y, vet[j].x));
23             j++;
24         }
25
26         auto it1 = s.lower_bound({vet[i].y - d, vet[i
       ].x});
27         auto it2 = s.upper_bound({vet[i].y + d, vet[i
       ].x});
28
29         for(auto it=it1; it!=it2; it++){
30             ll dx = vet[i].x - it->y;
31             ll dy = vet[i].y - it->x;
32
33             if(best_dist > dx*dx + dy*dy){
34                 best_dist = dx*dx + dy*dy;
35                 // closest pair points
36                 ans = mp(vet[i], point(it->y, it->x))
       ;
37             }
38         }
39
40         s.insert(point(vet[i].y, vet[i].x));
41     }
42
43     // best distance squared
44     return best_dist;
45 }
```

7.4 2d

```
1  #define vp vector<point>
2  #define ld long double
3  const ld EPS = 1e-6;
4  const ld PI = acos(-1);
```

```cpp
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+
       o.x, y+o.y}; }
17     point operator-(const point &o) const{ return {x-
       o.x, y-o.y}; }
18     point operator*(cod t) const{ return {x*t, y*t};
       }
19     point operator/(cod t) const{ return {x/t, y/t};
       }
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=
      collinear; 1=esq;
33     cod tmp = (b-a) ^ (e-a); // vector from a to b
34     return (tmp > EPS) - (tmp < -EPS);
35 }
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
      (a in front of b)
65     ld aux = angle(a,b)*180/PI;
66     return ((a^b)<=0 ? aux:360-aux);
67 }
68 bool angle_less(point a1, point b1, point a2, point
      b2){ // ang(a1,b1) <= ang(a2,b2)
69     point p1((a1*b1), abs((a1^b1)));
70     point p2((a2*b2), abs((a2^b2)));
71     return (p1^p2) <= 0;
72 }
73
74 ld area(vp &p){ // (points sorted)
75     ld ret = 0;
76     for(int i=2;i<(int)p.size();i++)
77         ret += (p[i]-p[0])^(p[i-1]-p[0]);
78     return abs(ret/2);
79 }
80 ld areaT(point &a, point &b, point &c){
81     return abs((b-a)^(c-a))/2.0;
82 }
83
84 point center(vp &A){
85     point c = point();
86     int len = A.size();
87     for(int i=0;i<len;i++)
88         c=c+A[i];
89     return c/len;
90 }
91
92 point forca_mod(point p, ld m){
93     ld cm = norm(p);
94     if(cm<EPS) return point();
95     return point(p.x*m/cm,p.y*m/cm);
96 }
97
98 ld param(point a, point b, point v){
99     // v = t*(b-a) + a // return t;
100    // assert(line(a, b).inside_seg(v));
101    return ((v-a) * (b-a)) / ((b-a) * (b-a));
102 }
103
104 bool simetric(vp &a){ //ordered
105    int n = a.size();
106    point c = center(a);
107    if(n&1) return false;
108    for(int i=0;i<n/2;i++)
109        if(ccw(a[i], a[i+n/2], c) != 0)
110            return false;
111    return true;
112 }
113
114 point mirror(point m1, point m2, point p){
115    // mirror point p around segment m1m2
116    point seg = m2-m1;
117    ld t0 = ((p-m1)*seg) / (seg*seg);
118    point ort = m1 + seg*t0;
119    point pm = ort-(p-ort);
120    return pm;
121 }
122
123
124 ///////////
125 //  Line  //
126 ///////////
127
128 struct line{
129    point p1, p2;
130    cod a, b, c; // ax+by+c = 0;
131    // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
132    line(point p1=0, point p2=0): p1(p1), p2(p2){
133        a = p1.y - p2.y;
134        b = p2.x - p1.x;
135        c = p1 ^ p2;
136    }
137    line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
       {
138        // Gera os pontos p1 p2 dados os coeficientes
139        // isso aqui eh um lixo mas quebra um galho
```

```
        kkkkkk
140         if(b==0){
141             p1 = point(1, -c/a);
142             p2 = point(0, -c/a);
143         }else{
144             p1 = point(1, (-c-a*1)/b);
145             p2 = point(0, -c/b);
146         }
147     }
148
149     cod eval(point p){
150         return a*p.x+b*p.y+c;
151     }
152     bool inside(point p){
153         return eq(eval(p), 0);
154     }
155     point normal(){
156         return point(a, b);
157     }
158
159     bool inside_seg(point p){
160         return (
161             ((p1-p) ^ (p2-p)) == 0 and
162             ((p1-p) * (p2-p)) <= 0
163         );
164     }
165 };
166
167
168 // be careful with precision error
169 vp inter_line(line l1, line l2){
170     ld det = l1.a*l2.b - l1.b*l2.a;
171     if(det==0) return {};
172     ld x = (l1.b*l2.c - l1.c*l2.b)/det;
173     ld y = (l1.c*l2.a - l1.a*l2.c)/det;
174     return {point(x, y)};
175 }
176
177 // segments not collinear
178 vp inter_seg(line l1, line l2){
179     vp ans = inter_line(l1, l2);
180     if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.
        inside_seg(ans[0]))
181         return {};
182     return ans;
183 }
184 bool seg_has_inter(line l1, line l2){
185     // if collinear
186     if (l1.inside_seg(l2.p1) || l1.inside_seg(l2.p2)
        || l2.inside_seg(l1.p1) || l2.inside_seg(l1.p2))
        return true;
187
188     return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.
        p2, l2.p2) < 0 and
189         ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.
        p2, l1.p2) < 0;
190 }
191
192 ld dist_seg(point p, point a, point b){ // point -
        seg
193     if((p-a)*(b-a) < EPS) return norm(p-a);
194     if((p-b)*(a-b) < EPS) return norm(p-b);
195     return abs((p-a)^(b-a)) / norm(b-a);
196 }
197
198 ld dist_line(point p, line l){ // point - line
199     return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
200 }
201
202 line bisector(point a, point b){
203     point d = (b-a)*2;
204     return line(d.x, d.y, a*a - b*b);
205 }
```

```
206
207 line perpendicular(line l, point p){ // passes
        through p
208     return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
209 }
210
211
212 ////////////
213 // Circle //
214 ////////////
215
216 struct circle{
217     point c; cod r;
218     circle() : c(0, 0), r(0){}
219     circle(const point o) : c(o), r(0){}
220     circle(const point a, const point b){
221         c = (a+b)/2;
222         r = norm(a-c);
223     }
224     circle(const point a, const point b, const point
        cc){
225         assert(ccw(a, b, cc) != 0);
226         c = inter_line(bisector(a, b), bisector(b, cc
        ))[0];
227         r = norm(a-c);
228     }
229     bool inside(const point &a) const{
230         return norm(a - c) <= r + EPS;
231     }
232 };
233
234 pair<point, point> tangent_points(circle cr, point p)
        {
235     ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
236     point p1 = rotccw(cr.c-p, -theta);
237     point p2 = rotccw(cr.c-p, theta);
238     assert(d1 >= cr.r);
239     p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
240     p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
241     return {p1, p2};
242 }
243
244
245 circle incircle(point p1, point p2, point p3){
246     ld m1 = norm(p2-p3);
247     ld m2 = norm(p1-p3);
248     ld m3 = norm(p1-p2);
249     point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
250     ld s = 0.5*(m1+m2+m3);
251     ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
252     return circle(c, r);
253 }
254
255 circle circumcircle(point a, point b, point c) {
256     circle ans;
257     point u = point((b-a).y, -(b-a).x);
258     point v = point((c-a).y, -(c-a).x);
259     point n = (c-b)*0.5;
260     ld t = (u^n)/(v^u);
261     ans.c = ((a+c)*0.5) + (v*t);
262     ans.r = norm(ans.c-a);
263     return ans;
264 }
265
266 vp inter_circle_line(circle C, line L){
267     point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.
        p1)*(ab) / (ab*ab));
268     ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s
         / (ab*ab);
269     if (h2 < -EPS) return {};
270     if (eq(h2, 0)) return {p};
271     point h = (ab/norm(ab)) * sqrt(h2);
272     return {p - h, p + h};
```

```
273 }
274
275 vp inter_circle(circle C1, circle C2){
276     if(C1.c == C2.c) { assert(C1.r != C2.r); return
        {}; }
277     point vec = C2.c - C1.c;
278     ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r
        ;
279     ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 =
        C1.r*C1.r - p*p*d2;
280     if (sum*sum < d2 or dif*dif > d2) return {};
281     point mid = C1.c + vec*p, per = point(-vec.y, vec
        .x) * sqrt(max((ld)0, h2) / d2);
282     if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
283     return {mid + per, mid - per};
284 }
285
286 // minimum circle cover O(n) amortizado
287 circle min_circle_cover(vp v){
288     random_shuffle(v.begin(), v.end());
289     circle ans;
290     int n = v.size();
291     for(int i=0;i<n;i++) if(!ans.inside(v[i])){
292         ans = circle(v[i]);
293         for(int j=0;j<i;j++) if(!ans.inside(v[j])){
294             ans = circle(v[i], v[j]);
295             for(int k=0;k<j;k++) if(!ans.inside(v[k])
        ){
296                 ans = circle(v[i], v[j], v[k]);
297             }
298         }
299     }
300     return ans;
301 }
```

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

8.5 Ternary Search

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

8.6 Binary Search First True

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

8.7   Biggest K

```
1  // Description: Gets sum of k biggest or k smallest
       elements in an array
2
3  // Problem: https://atcoder.jp/contests/abc306/tasks/
       abc306_e
4
5  // Complexity: O(log n)
6
7  struct SetSum {
8      ll s = 0;
9      multiset<ll> mt;
10     void add(ll x){
11         mt.insert(x);
12         s += x;
13     }
14     int pop(ll x){
15         auto f = mt.find(x);
16         if(f == mt.end()) return 0;
17         mt.erase(f);
18         s -= x;
19         return 1;
20     }
21 };
22
23 struct BigK {
24     int k;
25     SetSum gt, mt;
26     BigK(int _k){
27         k = _k;
28     }
29     void balancear(){
30         while((int)gt.mt.size() < k && (int)mt.mt.
    size()){
31             auto p = (prev(mt.mt.end()));
32             gt.add(*p);
33             mt.pop(*p);
34         }
35         while((int)mt.mt.size() && (int)gt.mt.size()
    &&
36         *(gt.mt.begin()) < *(prev(mt.mt.end())) ){
37             ll u = *(gt.mt.begin());
38             ll v = *(prev(mt.mt.end()));
39             gt.pop(u); mt.pop(v);
40             gt.add(v); mt.add(u);
41         }
42     }
43     void add(ll x){
44         mt.add(x);
45         balancear();
46     }
47     void rem(ll x){
48         //x = -x;
49         if(mt.pop(x) == 0)
50             gt.pop(x);
51         balancear();
52     }
53 };
```

```
54
55 int main() {
56     ios::sync_with_stdio(false);
57     cin.tie(NULL);
58
59     int n, k, q; cin >> n >> k >> q;
60
61     BigK big = BigK(k);
62
63     int arr[n] = {};
64
65     while (q--) {
66         int pos, num; cin >> pos >> num;
67         pos--;
68         big.rem(arr[pos]);
69         arr[pos] = num;
70         big.add(arr[pos]);
71
72         cout << big.gt.s << '\n';
73     }
74
75     return 0;
76 }
```

9   Data Structures

9.1   Sparse Table

```
1  // Description:
2  // Data structure to query for minimum and maximum
3
4  // Problem:
5  // https://cses.fi/problemset/task/1647/
6
7  // Complexity:
8  // Build O(n log n)
9  // Query O(1)
10
11 #include <bits/stdc++.h>
12
13 using namespace std;
14
15 const int MAX = 2e5+17;
16 const int INF = 1e9+17;
17
18 struct SparseTable {
19   int n;
20   vector<int> arr;
21   vector<vector<int>> st;
22   vector<int> log_2;
23
24   SparseTable(vector<int>& arr, int& n) : arr(arr), n
       (n) {
25     build();
26   }
27
28   void build() {
29     log_2.resize(MAX + 1);
30
31     log_2[1] = 0;
32     for (int i = 2; i <= MAX; i++) {
33       log_2[i] = log_2[i/2] + 1;
34     }
35
36     int K = log_2[n + 1];
37
38     st.resize(MAX, vector<int>(K + 1));
39
40     for (int i = 0; i < MAX; i++) {
41       for (int j = 0; j < K + 1; j++) {
42         st[i][j] = INF;
43       }
```

```
44      }
45
46      for (int i = 0; i < n; i++) {
47        st[i][0] = arr[i];
48      }
49
50      for (int j = 1; j <= K; j++) {
51        for (int i = 0; i + (1 << j) < MAX; i++) {
52          st[i][j] = min(st[i][j-1], st[i + (1 << (j -
      1))][j - 1]);
53        }
54      }
55    }
56
57    int query(int l, int r) {
58      int j = log_2[r - l + 1];
59      return min(st[l][j], st[r - (1 << j) + 1][j]);
60    }
61 };
```

9.2   Mergesort Tree Vector

```
1  // Description:
2  // In each node, the tree keeps a sorted list of
       elements in that range.
3  // It can be used to find how many elements are
       greater than x in a given range.
4  // It can also be used to find the position of an
       element if the list was sorted.
5  // query(i, j, k) - how many elements greater than k
       are in the range (i, j)
6
7  // Problem:
8  // https://www.spoj.com/problems/KQUERY
9
10 // Complexity:
11 // O(n log n) for build
12 // O(log ^ 2 n) for query
13
14 struct MergeSortTree {
15    vector<vector<int>> tree;
16    int n;
17
18    MergeSortTree(int n) : n(n) {
19      int sz = 1;
20      while (sz < n) sz *= 2;
21
22      tree.assign(2 * sz, vector<int>());
23    }
24
25    vector<int> merge(vector<int>& a, vector<int>& b) {
26      vector<int> res((int)a.size() + (int)b.size());
27      int it = 0, jt = 0, curr = 0;
28
29      while (it < (int)a.size() && jt < (int)b.size())
       {
30        if (a[it] <= b[jt]) {
31          res[curr++] = a[it++];
32        } else {
33          res[curr++] = b[jt++];
34        }
35      }
36
37      while (it < (int)a.size()) {
38        res[curr++] = a[it++];
39      }
40
41      while (jt < (int)b.size()) {
42        res[curr++] = b[jt++];
43      }
44
45      return res;
46    }
```

```
47
48    void build(int pos, int ini, int fim, vector<int>&
       v) {
49      if (ini == fim) {
50        if (ini < (int)v.size()) {
51          tree[pos].pb(v[ini]);
52        }
53        return;
54      }
55
56      int mid = ini + (fim - ini) / 2;
57
58      build(2 * pos + 1, ini, mid, v);
59      build(2 * pos + 2, mid + 1, fim, v);
60
61      tree[pos] = merge(tree[2 * pos + 1], tree[2 * pos
       + 2]);
62    }
63
64    // how many elements greater than val in vector v
65    int search(vector<int>& v, int val) {
66      auto it = upper_bound(v.begin(), v.end(), val);
67      if (it == v.end()) return 0;
68      return (int)v.size() - (it - v.begin());
69    }
70
71    // how many elements greater than val in the range
       (p, q)
72    int query(int pos, int ini, int fim, int p, int q,
       int val) {
73      if (fim < p || ini > q) {
74        return 0;
75      }
76
77      if (ini >= p && fim <= q) {
78        return search(tree[pos], val);
79      }
80
81      int mid = ini + (fim - ini) / 2;
82      return query(2 * pos + 1, ini, mid, p, q, val) +
       query(2 * pos + 2, mid + 1, fim, p, q, val);
83    }
84
85    void build(vector<int>& v) {
86      build(0, 0, n - 1, v);
87    }
88
89    int query(int p, int q, int val) {
90      return query(0, 0, n - 1, p, q, val);
91    }
92 };
```

9.3   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 }
```

9.4 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 // prioriy_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;
```

9.5 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 }
```

9.6 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 Programaçao da
      Unicamp 2023
8 // https://codeforces.com/group/WYIydkiPyE/contest
      /450037/attachments
9
10 // Complexity:
11 // Add and remove elements - O(log n)
```

```cpp
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    }
```

```cpp
83
84    ll sum_big() {
85      return sumb;
86    }
87
88    int median() {
89      return *big.begin();
90    }
91  };
```

9.7 Psum2d

```cpp
1  // Description:
2  // Queries the sum of a rectangle that goes from grid
        [from_row][from_col] to grid[to_row][to_col]
3
4  // Problem:
5  // https://cses.fi/problemset/task/1652/
6
7  // Complexity:
8  // O(n) build
9  // O(1) query
10
11  for (int i = 1; i <= n; i++) {
12    for (int j = 1; j <= n; j++) {
13      psum[i][j] = grid[i][j] + psum[i - 1][j] + psum[i
        ][j - 1] - psum[i - 1][j - 1];
14    }
15  }
16
17  while (q--) {
18    int from_row, to_row, from_col, to_col;
19    cin >> from_row >> from_col >> to_row >> to_col;
20    cout << psum[to_row][to_col] - psum[from_row - 1][
        to_col] -
21    psum[to_row][from_col - 1] + psum[from_row - 1][
        from_col - 1] << '\n';
22  }
```

9.8 Dynamic Implicit Sparse

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

35

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

9.9 Segtree2d

```
1  // Description:
2  // Indexed at zero
3  // Given a N x M grid, where i represents the row and
       j the column, perform the following operations
4  // update(i, j) - update the value of grid[i][j]
5  // query(i1, j1, i2, j2) - return the sum of values
       inside the rectangle
6  // defined by grid[i1][j1] and grid[i2][j2] inclusive
7
8  // Problem:
9  // 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, m);
20 // vector<vector<int>> v(n, vector<int>(m));
21 // seg.build(v);
22
23 struct Segtree2D {
24     const int MAXN = 1025;
25     const int NEUTRAL = 0;
26     int N, M;
27
28     vector<vector<int>> seg;
29
30     Segtree2D(int N, int M) {
31         this->N = N;
32         this->M = M;
33         seg.assign(4*MAXN, vector<int>(4*MAXN,
    NEUTRAL));
34     }
35
36     int f(int a, int b) {
37       return max(a, b);
38     }
39
40     void buildY(int noX, int lX, int rX, int noY, int
     lY, int rY, vector<vector<int>> &v){
41         if(lY == rY){
42             if(lX == rX){
43                 seg[noX][noY] = v[rX][rY];
44             }else{
45                 seg[noX][noY] = f(seg[2*noX+1][noY],
    seg[2*noX+2][noY]);
46             }
47         }else{
48             int m = (lY+rY)/2;
49
50             buildY(noX, lX, rX, 2*noY+1, lY, m, v);
51             buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);
52
53             seg[noX][noY] = f(seg[noX][2*noY+1], seg[
    noX][2*noY+2]);
54         }
55     }
56
57     void buildX(int noX, int lX, int rX, vector<
    vector<int>> &v){
58         if(lX != rX){
59             int m = (lX+rX)/2;
60
61             buildX(2*noX+1, lX, m, v);
62             buildX(2*noX+2, m+1, rX, v);
63         }
64
65         buildY(noX, lX, rX, 0, 0, M - 1, v);
66     }
67
68     void updateY(int noX, int lX, int rX, int noY,
    int lY, int rY, int y){
69         if(lY == rY){
```

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

```
128     int query(int aX, int aY, int bX, int bY) {
129         return queryX(0, 0, N - 1, aX, bX, aY, bY);
130     }
131
132     void update(int x, int y) {
133         updateX(0, 0, N - 1, x, y);
134     }
135 };
```

9.10 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
        q) {
46         if (ini >= p && fim <= q) {
47             return seg[pos];
48         }
49
50         if (q < ini || p > fim) {
51             return NEUTRAL;
52         }
53
54         int e = 2*pos + 1;
55         int d = 2*pos + 2;
56         int m = ini + (fim - ini) / 2;
57
```

```
58          return f(query(e, ini, m, p, q), query(d, m +
     1, fim, p, q));
59      }
60
61      void update(int pos, int ini, int fim, int id,
     int val) {
62          if (ini > id || fim < id) {
63              return;
64          }
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>
     &v) {
83          if (ini == fim) {
84              if (ini < (int)v.size()) {
85                  seg[pos] = mp(v[ini], 1);
86              }
87              return;
88          }
89
90          int e = 2*pos + 1;
91          int d = 2*pos + 2;
92          int m = ini + (fim - ini) / 2;
93
94          build(e, ini, m, v);
95          build(d, m + 1, fim, v);
96
97          seg[pos] = f(seg[e], seg[d]);
98      }
99
100     ftype query(int p, int q) {
101         return query(0, 0, n - 1, p, q);
102     }
103
104     void update(int id, int val) {
105         update(0, 0, n - 1, id, val);
106     }
107
108     void build(vector<int> &v) {
109         build(0, 0, n - 1, v);
110     }
111
112     void debug() {
113         for (auto e : seg) {
114             cout << e.ff << ' ' << e.ss << '\n';
115         }
116         cout << '\n';
117     }
118 };
```

9.11   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         if (q < ini || p > fim) {
73             return NEUTRAL;
74         }
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 };
```

9.12   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 + 3 =
       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(0LL, 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,
       ftype_node suf, ftype_node sum) : max_seg(max_seg
       ), pref(pref), suf(suf), sum(sum) {};
35 };
36
37 typedef Node ftype;
38
39 struct Segtree {
40     vector<ftype> seg;
41     int n;
42     const ftype NEUTRAL = Node(0, 0, 0, 0);
43
44     Segtree(int n) {
45         int sz = 1;
46         // potencia de dois mais proxima
47         while (sz < n) sz *= 2;
48         this->n = sz;
49
50         // numero de nos da seg
51         seg.assign(2*sz, NEUTRAL);
52     }
```

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

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

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

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

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

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

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

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

            return;
        }

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

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

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

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

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

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

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

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

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

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

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

9.13 Range Query Point Update

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

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

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

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

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

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

typedef long long ftype;

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

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

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

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

```
43    ftype query(int pos, int ini, int fim, int p, int
       q) {
44        if (ini >= p && fim <= q) {
45            return seg[pos];
46        }
47
48        if (q < ini || p > fim) {
49            return NEUTRAL;
50        }
51
52        int e = 2*pos + 1;
53        int d = 2*pos + 2;
54        int m = ini + (fim - ini) / 2;
55
56        return f(query(e, ini, m, p, q), query(d, m +
       1, fim, p, q));
57    }
58
59    void update(int pos, int ini, int fim, int id,
      int val) {
60        if (ini > id || fim < id) {
61            return;
62        }
63
64        if (ini == id && fim == id) {
65            seg[pos] = val;
66
67            return;
68        }
69
70        int e = 2*pos + 1;
71        int d = 2*pos + 2;
72        int m = ini + (fim - ini) / 2;
73
74        update(e, ini, m, id, val);
75        update(d, m + 1, fim, id, val);
76
77        seg[pos] = f(seg[e], seg[d]);
78    }
79
80    void build(int pos, int ini, int fim, vector<int>
       &v) {
81        if (ini == fim) {
82            if (ini < (int)v.size()) {
83                seg[pos] = v[ini];
84            }
85            return;
86        }
87
88        int e = 2*pos + 1;
89        int d = 2*pos + 2;
90        int m = ini + (fim - ini) / 2;
91
92        build(e, ini, m, v);
93        build(d, m + 1, fim, v);
94
95        seg[pos] = f(seg[e], seg[d]);
96    }
97
98    ftype query(int p, int q) {
99        return query(0, 0, n - 1, p, q);
100   }
101
102   void update(int id, int val) {
103        update(0, 0, n - 1, id, val);
104   }
105
106   void build(vector<int> &v) {
107        build(0, 0, n - 1, v);
108   }
109
110   void debug() {
111        for (auto e : seg) {
112            cout << e << ' ';
113        }
114        cout << '\n';
115   }
116 };
```

9.14   Lazy Assignment To Segment

```
1  const long long INF = 1e18+10;
2
3  typedef long long ftype;
4
5  struct Segtree {
6      vector<ftype> seg;
7      vector<ftype> lazy;
8      int n;
9      const ftype NEUTRAL = 0;
10     const ftype NEUTRAL_LAZY = -1; // Change to -INF
      if there are negative numbers
11
12     Segtree(int n) {
13         int sz = 1;
14         // potencia de dois mais proxima
15         while (sz < n) sz *= 2;
16         this->n = sz;
17
18         // numero de nos da seg
19         seg.assign(2*sz, NEUTRAL);
20         lazy.assign(2*sz, NEUTRAL_LAZY);
21     }
22
23     ftype apply_lazy(ftype a, ftype b, int len) {
24         if (b == NEUTRAL_LAZY) return a;
25         if (a == NEUTRAL_LAZY) return b * len;
26         else return b * len;
27     }
28
29     void propagate(int pos, int ini, int fim) {
30         if (ini == fim) {
31             return;
32         }
33
34         int e = 2*pos + 1;
35         int d = 2*pos + 2;
36         int m = ini + (fim - ini) / 2;
37
38         lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
39         lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);
40
41         seg[e] = apply_lazy(seg[e], lazy[pos], m -
      ini + 1);
42         seg[d] = apply_lazy(seg[d], lazy[pos], fim -
      m);
43
44         lazy[pos] = NEUTRAL_LAZY;
45     }
46
47     ftype f(ftype a, ftype b) {
48         return a + b;
49     }
50
51     ftype query(int pos, int ini, int fim, int p, int
       q) {
52         propagate(pos, ini, fim);
53
54         if (ini >= p && fim <= q) {
55             return seg[pos];
56         }
57
58         if (q < ini || p > fim) {
59             return NEUTRAL;
60         }
61
```

```
62          int e = 2*pos + 1;
63          int d = 2*pos + 2;
64          int m = ini + (fim - ini) / 2;
65
66          return f(query(e, ini, m, p, q), query(d, m +
     1, fim, p, q));
67      }
68
69      void update(int pos, int ini, int fim, int p, int
     q, int val) {
70          propagate(pos, ini, fim);
71
72          if (ini > q || fim < p) {
73              return;
74          }
75
76          if (ini >= p && fim <= q) {
77              lazy[pos] = apply_lazy(lazy[pos], val, 1)
     ;
78              seg[pos] = apply_lazy(seg[pos], val, fim
     - ini + 1);
79
80              return;
81          }
82
83          int e = 2*pos + 1;
84          int d = 2*pos + 2;
85          int m = ini + (fim - ini) / 2;
86
87          update(e, ini, m, p, q, val);
88          update(d, m + 1, fim, p, q, val);
89
90          seg[pos] = f(seg[e], seg[d]);
91      }
92
93      void build(int pos, int ini, int fim, vector<int>
     &v) {
94          if (ini == fim) {
95              // se a çãposio existir no array original
96              // seg tamanho potencia de dois
97              if (ini < (int)v.size()) {
98                  seg[pos] = v[ini];
99              }
100             return;
101         }
102
103         int e = 2*pos + 1;
104         int d = 2*pos + 2;
105         int m = ini + (fim - ini) / 2;
106
107         build(e, ini, m, v);
108         build(d, m + 1, fim, v);
109
110         seg[pos] = f(seg[e], seg[d]);
111     }
112
113     ftype query(int p, int q) {
114         return query(0, 0, n - 1, p, q);
115     }
116
117     void update(int p, int q, int val) {
118         update(0, 0, n - 1, p, q, val);
119     }
120
121     void build(vector<int> &v) {
122         build(0, 0, n - 1, v);
123     }
124
125     void debug() {
126         for (auto e : seg) {
127             cout << e << ' ';
128         }
129         cout << '\n';
```

```
130         for (auto e : lazy) {
131             cout << e << ' ';
132         }
133         cout << '\n';
134         cout << '\n';
135     }
136 };
```

9.15 Lazy Dynamic Implicit Sparse

```
1 // Description:
2 // Indexed at one
3
4 // When the indexes of the nodes are too big to be
       stored in an array
5 // and the queries need to be answered online so we
       can't sort the nodes and compress them
6 // we create nodes only when they are needed so there
       'll be (Q*log(MAX)) nodes
7 // where Q is the number of queries and MAX is the
       maximum index a node can assume
8
9 // Query - get sum of elements from range (l, r)
       inclusive
10 // Update - update element at position id to a value
       val
11
12 // Problem:
13 // https://oj.uz/problem/view/IZhO12_apple
14
15 // Complexity:
16 // O(log n) for both query and update
17
18 // How to use:
19 // MAX is the maximum index a node can assume
20 // Create a default null node
21 // Create a node to be the root of the segtree
22
23 // Segtree seg = Segtree(MAX);
24
25 const int MAX = 1e9+10;
26 const long long INF = 1e18+10;
27
28 typedef long long ftype;
29
30 struct Segtree {
31     vector<ftype> seg, d, e, lazy;
32     const ftype NEUTRAL = 0;
33     const ftype NEUTRAL_LAZY = -1; // change to -INF
       if the elements can be negative
34     int n;
35
36     Segtree(int n) {
37         this->n = n;
38         create();
39         create();
40     }
41
42     ftype apply_lazy(ftype a, ftype b, int len) {
43         if (b == NEUTRAL_LAZY) return a;
44         else return b * len; // change to a + b * len
       to add to an element instead of updating it
45     }
46
47     void propagate(int pos, int ini, int fim) {
48         if (seg[pos] == 0) return;
49
50         if (ini == fim) {
51             return;
52         }
53
54         int m = (ini + fim) >> 1;
55
```

```
56          if (e[pos] == 0) e[pos] = create();
57          if (d[pos] == 0) d[pos] = create();
58
59          lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[
   pos], 1);
60          lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[
   pos], 1);
61
62          seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[
   pos], m - ini + 1);
63          seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[
   pos], fim - m);
64
65          lazy[pos] = NEUTRAL_LAZY;
66      }
67
68      ftype f(ftype a, ftype b) {
69          return a + b;
70      }
71
72      ftype create() {
73          seg.push_back(0);
74          e.push_back(0);
75          d.push_back(0);
76          lazy.push_back(-1);
77          return seg.size() - 1;
78      }
79
80      ftype query(int pos, int ini, int fim, int p, int
    q) {
81          propagate(pos, ini, fim);
82          if (q < ini || p > fim) return NEUTRAL;
83          if (pos == 0) return 0;
84          if (p <= ini && fim <= q) return seg[pos];
85          int m = (ini + fim) >> 1;
86          return f(query(e[pos], ini, m, p, q), query(d
   [pos], m + 1, fim, p, q));
87      }
88
89      void update(int pos, int ini, int fim, int p, int
    q, int val) {
90          propagate(pos, ini, fim);
91          if (ini > q || fim < p) {
92              return;
93          }
94
95          if (ini >= p && fim <= q) {
96              lazy[pos] = apply_lazy(lazy[pos], val, 1)
   ;
97              seg[pos] = apply_lazy(seg[pos], val, fim
   - ini + 1);
98
99              return;
100         }
101
102         int m = (ini + fim) >> 1;
103
104         if (e[pos] == 0) e[pos] = create();
105         update(e[pos], ini, m, p, q, val);
106
107         if (d[pos] == 0) d[pos] = create();
108         update(d[pos], m + 1, fim, p, q, val);
109
110         seg[pos] = f(seg[e[pos]], seg[d[pos]]);
111     }
112
113     ftype query(int p, int q) {
114         return query(1, 1, n, p, q);
115     }
116
117     void update(int p, int q, int val) {
118         update(1, 1, n, p, q, val);
119     }
```

```
120 };
```

9.16 Persistent

```
1 // Description:
2 // Persistent segtree allows for you to save the
     different versions of the segtree between each
     update
3 // Indexed at one
4 // Query - get sum of elements from range (l, r)
     inclusive
5 // Update - update element at position id to a value
     val
6
7 // Problem:
8 // https://cses.fi/problemset/task/1737/
9
10 // Complexity:
11 // O(log n) for both query and update
12
13 // How to use:
14 // vector<int> raiz(MAX); // vector to store the
     roots of each version
15 // Segtree seg = Segtree(INF);
16 // raiz[0] = seg.create(); // null node
17 // curr = 1; // keep track of the last version
18
19 // raiz[k] = seg.update(raiz[k], idx, val); //
     updating version k
20 // seg.query(raiz[k], l, r) // querying version k
21 // raiz[++curr] = raiz[k]; // create a new version
     based on version k
22
23 const int MAX = 2e5+17;
24 const int INF = 1e9+17;
25
26 typedef long long ftype;
27
28 struct Segtree {
29     vector<ftype> seg, d, e;
30     const ftype NEUTRAL = 0;
31     int n;
32
33     Segtree(int n) {
34         this->n = n;
35     }
36
37     ftype f(ftype a, ftype b) {
38         return a + b;
39     }
40
41     ftype create() {
42         seg.push_back(0);
43         e.push_back(0);
44         d.push_back(0);
45         return seg.size() - 1;
46     }
47
48     ftype query(int pos, int ini, int fim, int p, int
    q) {
49         if (q < ini || p > fim) return NEUTRAL;
50         if (pos == 0) return 0;
51         if (p <= ini && fim <= q) return seg[pos];
52         int m = (ini + fim) >> 1;
53         return f(query(e[pos], ini, m, p, q), query(d
   [pos], m + 1, fim, p, q));
54     }
55
56     int update(int pos, int ini, int fim, int id, int
    val) {
57         int novo = create();
58
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 };
```

9.17 Sparse Table2d

```
1 // Description
2 // Minimum queries in a 2D grid
3
4 // Problem:
5 // https://codeforces.com/group/YgJmumGtHD/contest
      /103794/problem/D
6
7 // Complexity:
8 // Build O(N * M * log(N) * log(M))
9 // Query O(1)
10 // Memory COmplexity: O(N * M * log(N) * log(M))
11
12 const int MAX = 410;
13
14 struct SparseTable2D {
15   vector<vector<int>> matrix;
16   vector<vector<vector<vector<int>>>> table;
17   int n, m;
18
19   SparseTable2D(vector<vector<int>>& matrix, int n,
      int m) : matrix(matrix), n(n), m(m) {
20     table.resize(MAX, vector<vector<vector<int>>>(MAX
      , vector<vector<int>>(log2(MAX) + 1, vector<int>(
      log2(MAX) + 1))));
21     build();
22   }
23
24   int f(int a, int b) {
25     return max(a, b);
26   }
27
28   void build() {
29     for (int i = 0; i < n; i++) {
30       for (int j = 0; j < m; j++) {
31         table[i][j][0][0] = matrix[i][j];
32       }
33     }
34
35     for (int k = 1; k <= (int)(log2(n)); k++) {
36       for (int i = 0; i + (1 << k) - 1 < n; i++) {
37         for (int j = 0; j + (1 << k) - 1 < m; j++) {
38           table[i][j][k][0] = f(
39           table[i][j][k - 1][0],
40           table[i + (1 << (k - 1))][j][k - 1][0]);
41         }
42       }
43     }
44
45     for (int k = 1; k <= (int)(log2(m)); k++) {
46       for (int i = 0; i < n; i++) {
47         for (int j = 0; j + (1 << k) - 1 < m; j++) {
48           table[i][j][0][k] = f(
49           table[i][j][0][k - 1],
50           table[i][j + (1 << (k - 1))][0][k - 1]);
51         }
52       }
53     }
54
55     for (int k = 1; k <= (int)(log2(n)); k++) {
56       for (int l = 1; l <= (int)(log2(m)); l++) {
57         for (int i = 0; i + (1 << k) - 1 < n; i++) {
58           for (int j = 0; j + (1 << l) - 1 < m; j++)
   {
59             table[i][j][k][l] = f(
60               f(
61                 table[i][j][k - 1][l - 1],
62                 table[i + (1 << (k - 1))][j][k - 1][l
   - 1]
63               ),
64               f(
65                 table[i][j + (1 << (l - 1))][k - 1][l
   - 1],
66                 table[i + (1 << (k - 1))][j + (1 << (
   l - 1))][k - 1][l - 1])
67               );
68           }
69         }
70       }
71     }
72   }
73
74   int query(int x1, int y1, int x2, int y2) {
75     int k = log2(x2 - x1 + 1);
76     int l = log2(y2 - y1 + 1);
77
78     return f(
79       f(
80         table[x1][y1][k][l],
81         table[x2 - (1 << k) + 1][y1][k][l]
82       ),
83       f(
84         table[x1][y2 - (1 << l) + 1][k][l],
85         table[x2 - (1 << k) + 1][y2 - (1 << l) + 1][k
   ][l]
86       )
87     );
88   }
89 };
```