Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1 Math **2**
1.1 Ceil . . . 2
1.2 To Decimal . . . 2
1.3 Matrix Exponentiation . . . 2
1.4 Crt . . . 3
1.5 Binary To Decimal . . . 3
1.6 Fast Exponentiation . . . 3
1.7 Linear Diophantine Equation . . . 3
1.8 Function Root . . . 4
1.9 Sieve Of Eratosthenes . . . 4
1.10 Horner Algorithm . . . 4
1.11 Multiplicative Inverse . . . 4
1.12 Representation Arbitrary Base . . . 4
1.13 Set Operations . . . 5
1.14 Divisors . . . 5
1.15 Check If Bit Is On . . . 5
1.16 Prime Factors . . . 5

2 DP **5**
2.1 Knapsack With Index . . . 5
2.2 Substr Palindrome . . . 5
2.3 Edit Distance . . . 6
2.4 Knapsack . . . 6
2.5 Digits . . . 6
2.6 Coins . . . 6
2.7 Minimum Coin Change . . . 6
2.8 Kadane . . . 7

3 Template **7**
3.1 Template . . . 7
3.2 Template Clean . . . 7

4 Strings **7**
4.1 Kmp . . . 7
4.2 Generate All Permutations . . . 8
4.3 Generate All Sequences Length K . . . 8
4.4 Lcs . . . 8
4.5 Trie . . . 8
4.6 Z-function . . . 9

5 Misc **9**
5.1 Split . . . 9
5.2 Int128 . . . 9

6 Graphs **9**
6.1 Centroid Find . . . 9
6.2 Bipartite . . . 9
6.3 Prim . . . 10
6.4 Ford Fulkerson Edmonds Karp . . . 10
6.5 Floyd Warshall . . . 10
6.6 Lca . . . 11
6.7 Bellman Ford . . . 12
6.8 Dinic . . . 12
6.9 2sat . . . 13
6.10 Find Cycle . . . 15
6.11 Cycle Path Recovery . . . 15
6.12 Centroid Decomposition . . . 16
6.13 Tarjan Bridge . . . 16
6.14 Small To Large . . . 17
6.15 Tree Diameter . . . 17
6.16 Dijkstra . . . 17
6.17 Kruskall . . . 18

7 Geometry **18**
7.1 2d . . . 18

8 Algorithms **21**
8.1 Lis 21
8.2 Delta-encoding 21
8.3 Subsets 21
8.4 Binary Search Last True 21
8.5 Ternary Search 21
8.6 Binary Search First True 21
8.7 Biggest K 21

9 Data Structures **22**
9.1 Ordered Set 22
9.2 Priority Queue 22
9.3 Dsu 23
9.4 Two Sets 23
9.5 Dynamic Implicit Sparse 24
9.6 Segtree2d 24
9.7 Minimum And Amount 26
9.8 Lazy Addition To Segment 26
9.9 Segment With Maximum Sum 28
9.10 Range Query Point Update 29
9.11 Lazy Assignment To Segment 29
9.12 Lazy Dynamic Implicit Sparse 30
9.13 Persistent 31

1 Math

1.1 Ceil

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

1.2 To Decimal

```cpp
const string digits { "0123456789
    ABCDEFGHIJKLMNOPQRSTUVWXYZ" };

long long to_decimal(const string& rep, long long
    base) {
  long long n = 0;

  for (auto c : rep) {
    n *= base;
    n += digits.find(c);
  }

  return n;
}
```

1.3 Matrix Exponentiation

```cpp
// Description:
// Calculate the nth term of a linear recursion

// Example Fibonacci:
// Given a linear recurrence, for example fibonacci
// F(n) = n, x <= 1
// F(n) = F(n - 1) + F(n - 2), x > 1

// The recurrence has two terms, so we can build a
    matrix 2 x 1 so that
// n + 1 = transition * n

// (2 x 1) = (2 x 2) * (2 x 1)
// F(n)      = a b * F(n - 1)
// F(n - 1)    c d   F(n - 2)

// Another Example:
// Given a grid 3 x n, you want to color it using 3
    distinct colors so that
// no adjacent place has the same color. In how many
    different ways can you do that?
// There are 6 ways for the first column to be
    colored using 3 distinct colors
// ans 6 ways using 2 equal colors and 1 distinct one

// Adding another column, there are:
// 3 ways to go from 2 equal to 2 equal
// 2 ways to go from 2 equal to 3 distinct
// 2 ways to go from 3 distinct to 2 equal
// 2 ways to go from 3 distinct to 3 distinct

// So we star with matrix 6 6 and multiply it by the
    transition 3 2 and get 18 12
//                         6 6
//                 2 2       12 12
// the we can exponentiate this matrix to find the
    nth column

// Problem:
// https://cses.fi/problemset/task/1722/

// Complexity:
// O(log n)

// How to use:
// vector<vector<ll>> v = {{1, 1}, {1, 0}};
// Matriz transition = Matriz(v);
// cout << fexp(transition, n)[0][1] << '\n';

using ll = long long;

const int MOD = 1e9+7;

struct Matriz{
    vector<vector<ll>> mat;
    int rows, columns;

    vector<ll> operator[](int i){
        return mat[i];
    }

    Matriz(vector<vector<ll>>& matriz){
        mat = matriz;
        rows = mat.size();
        columns = mat[0].size();
    }

    Matriz(int row, int column, bool identity=false){
        rows = row;  columns = column;
        mat.assign(rows, vector<ll>(columns, 0));
        if(identity) {
            for(int i = 0; i < min(rows,columns); i
    ++) {
                mat[i][i] = 1;
            }
        }
    }

    Matriz operator * (Matriz a) {
        assert(columns == a.rows);
        vector<vector<ll>> resp(rows, vector<ll>(a.
    columns, 0));

        for(int i = 0; i < rows; i++){
            for(int j = 0; j < a.columns; j++){
                for(int k = 0; k < a.rows; k++){
                    resp[i][j] = (resp[i][j] + (mat[i
    ][k] * 1LL * a[k][j]) % MOD) % MOD;
                }
            }
        }
        return Matriz(resp);
    }

    Matriz operator + (Matriz a) {
        assert(rows == a.rows && columns == a.columns
    );
        vector<vector<ll>> resp(rows, vector<ll>(
    columns,0));
        for(int i = 0; i < rows; i++){
            for(int j = 0; j < columns; j++){
                resp[i][j] = (resp[i][j] + mat[i][j]
    + a[i][j]) % MOD;
            }
        }
        return Matriz(resp);
    }
};

Matriz fexp(Matriz base, ll exponent){
    Matriz result = Matriz(base.rows, base.rows, 1);
    while(exponent > 0){
        if(exponent & 1LL) result = result * base;
        base = base * base;
        exponent  = exponent >> 1;
    }
    return result;
```

```
}
```

1.4 Crt

```
ll crt(const vector<pair<ll, ll>> &vet){
    ll ans = 0, lcm = 1;
    ll a, b, g, x, y;
    for(const auto &p : vet) {
        tie(a, b) = p;
        tie(g, x, y) = gcd(lcm, b);
        if((a - ans) % g != 0) return -1; // no solution
        ans = ans + x * ((a - ans) / g) % (b / g) * lcm;
        lcm = lcm * (b / g);
        ans = (ans % lcm + lcm) % lcm;
    }
    return ans;
}
```

1.5 Binary To Decimal

```
int binary_to_decimal(long long n) {
  int dec = 0, i = 0, rem;

  while (n!=0) {
    rem = n % 10;
    n /= 10;
    dec += rem * pow(2, i);
    ++i;
  }

  return dec;
}

long long decimal_to_binary(int n) {
  long long bin = 0;
  int rem, i = 1;

  while (n!=0) {
    rem = n % 2;
    n /= 2;
    bin += rem * i;
    i *= 10;
  }

  return bin;
}
```

1.6 Fast Exponentiation

```
ll fexp(ll b, ll e, ll mod) {
    ll res = 1;
    b %= mod;
    while(e){
        if(e & 1LL)
            res = (res * b) % mod;
        e = e >> 1LL;
        b = (b * b) % mod;
    }
    return res;
}
```

1.7 Linear Diophantine Equation

```
// int a, b, c, x1, x2, y1, y2; cin >> a >> b >> c >> x1 >> x2 >> y1 >> y2;
// int ans = -1;
// if (a == 0 && b == 0) {
//     if (c != 0) ans = 0;
//     else ans = (x2 - x1 + 1) * (y2 - y1 + 1);
// }
// else if (a == 0) {
//     if (c % b == 0 && y1 <= c / b && y2 >= c / b) ans = (x2 - x1 + 1);
//     else ans = 0;
// }
// else if (b == 0) {
//     if (c % a == 0 && x1 <= c / a && x2 >= c / a) ans = (y2 - y1 + 1);
//     else ans = 0;
// }

// Careful when a or b are negative or zero

// if (ans == -1) ans =  find_all_solutions(a, b, c, x1, x2, y1, y2);
// cout << ans << '\n';

// Problems:
// https://www.spoj.com/problems/CEQU/
// http://codeforces.com/problemsets/acmsguru/problem/99999/106

// consider trivial case a or b is 0
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

// x and y are one solution and g is the gcd, all passed as reference
// minx <= x <= maxx miny <= y <= maxy
bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

// return number of solutions in the interval
int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
```

```
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

1.8 Function Root

```
const ld EPS1 = 1e-9; // iteration precision error
const ld EPS2 = 1e-4; // output precision error

ld f(ld x) {
  // exp(-x) == e^(-x)
  return p * exp(-x) + q * sin(x) + r * cos(x) + s *
    tan(x) + t * x * x + u;
}

ld root(ld a, ld b) {
  while (b - a >= EPS1) {
    ld c = (a + b) / 2.0;
    ld y = f(c);

    if (y < 0) b = c;
    else a = c;
  }

  return (a + b) / 2;
}

int main() {
  ld ans = root(0, 1);
  if (abs(f(ans)) <= EPS2) cout << fixed <<
    setprecision(4) << ans << '\n';
  else  cout << "No solution\n";

  return 0;
}
```

1.9 Sieve Of Eratosthenes

```
vector<bool> is_prime(MAX, true);
vector<int> primes;

void sieve() {
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i < MAX; i++) {
        if (is_prime[i]) {
            primes.push_back(i);

            for (int j = i + i; j < MAX; j += i)
                is_prime[j] = false;
        }
    }
}
```

1.10 Horner Algorithm

```
// Description:
// Evaluates y = f(x)

// Problem:
// https://onlinejudge.org/index.php?option=
    com_onlinejudge&Itemid=8&page=show_problem&
    problem=439

// Complexity:
// O(n)

using polynomial = std::vector<int>;

polynomial p {6, -5, 2}; // p(x) = x^2 - 5x + 6;

int degree(const polynomial& p) {
  return p.size() - 1;
}

int evaluate(const polynomial& p, int x) {
  int y = 0, N = degree(p);

  for (int i = N; i >= 0; --i) {
    y *= x;
    y += p[i];
  }

  return y;
}
```

1.11 Multiplicative Inverse

```
ll extend_euclid(ll a, ll b, ll &x, ll &y) {
    if (a == 0)
    {
        x = 0; y = 1;
        return b;
    }
    ll x1, y1;
    ll d = extend_euclid(b%a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}

// gcd(a, m) = 1 para existir solucao
// ax + my = 1, ou a*x = 1 (mod m)
ll inv_gcd(ll a, ll m) { // com gcd
  ll x, y;
  extend_euclid(a, m, x, y);
  return (((x % m) +m) %m);
}

ll inv(ll a, ll phim) { // com phi(m), se m for primo
    entao phi(m) = p-1
  ll e = phim-1;
  return fexp(a, e, MOD);
}
```

1.12 Representation Arbitrary Base

```
const string digits { "0123456789
    ABCDEFGHIJKLMNOPQRSTUVWXYZ" };

string representation(int n, int b) {
  string rep;

  do {
    rep.push_back(digits[n % b]);
    n /= b;
  } while (n);

  reverse(rep.begin(), rep.end());

  return rep;
}
```

1.13 Set Operations

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

vector<int> res;
set_union(s1.begin(), s1.end(), s2.begin(), s2.end(),
     inserter(res, res.begin()));
set_intersection(s1.begin(), s1.end(), s2.begin(), s2
    .end(), inserter(res, res.begin()));
// present in the first set, but not in the second
set_difference(s1.begin(), s1.end(), s2.begin(), s2.
    end(), inserter(res, res.begin()));
// present in one of the sets, but not in the other
set_symmetric_difference(s1.begin(), s1.end(), s2.
    begin(), s2.end(), inserter(res, res.begin()));
```

1.14 Divisors

```
vector<long long> all_divisors(long long n) {
  vector<long long> ans;
  for(long long a = 1; a*a <= n; a++){
    if(n % a == 0) {
      long long b = n / a;
      ans.push_back(a);
      if(a != b) ans.push_back(b);
    }
  }
  sort(ans.begin(), ans.end());
  return ans;
}
```

1.15 Check If Bit Is On

```
// msb de 0 é undefined
#define msb(n) (32 - __builtin_clz(n))
// #define msb(n) (64 - __builtin_clzll(n) )
// popcount
// turn bit off

bool bit_on(int n, int bit) {
    if(1 & (n >> bit)) return true;
    else return false;
}
```

1.16 Prime Factors

```
vector<pair<long long, int>> fatora(long long n) {
  vector<pair<long long, int>> ans;
  for(long long p = 2; p*p <= n; p++) {
    if(n % p == 0) {
      int expoente = 0;
      while(n % p == 0) {
        n /= p;
        expoente++;
      }
      ans.emplace_back(p, expoente);
    }
  }
  if(n > 1) ans.emplace_back(n, 1);
  return ans;
}
```

2 DP

2.1 Knapsack With Index

```
void knapsack(int W, int wt[], int val[], int n) {
    int i, w;
    int K[n + 1][W + 1];

    for (i = 0; i <= n; i++) {
        for (w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] +
                    K[i - 1][w - wt[i - 1]], K[i -
    1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }

    int res = K[n][W];
    cout<< res << endl;

    w = W;
    for (i = n; i > 0 && res > 0; i--) {
        if (res == K[i - 1][w])
            continue;
        else {
            cout<<" "<<wt[i - 1] ;
            res = res - val[i - 1];
            w = w - wt[i - 1];
        }
    }
}

int main()
{
    int val[] = { 60, 100, 120 };
    int wt[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);

    knapsack(W, wt, val, n);

    return 0;
}
```

2.2 Substr Palindrome

```
// êvoc deve informar se a substring de S formada
    pelos elementos entre os indices i e j
// é um palindromo ou ãno.

char s[MAX];
int calculado[MAX][MAX]; // inciado com false, ou 0
int tabela[MAX][MAX];

int is_palin(int i, int j){
  if(calculado[i][j]){
    return tabela[i][j];
```

```
  }
  if(i == j) return true;
  if(i + 1 == j) return s[i] == s[j];

  int ans = false;
  if(s[i] == s[j]){
    if(is_palin(i+1, j-1)){
      ans = true;
    }
  }
  calculado[i][j] = true;
  tabela[i][j] = ans;
  return ans;
}
```

2.3 Edit Distance

```
// Description:
// Minimum number of operations required to transform a string into another
// Operations allowed: add character, remove character, replace character

// Parameters:
// str1 - string to be transformed into str2
// str2 - string that str1 will be transformed into
// m - size of str1
// n - size of str2

// Problem:
// https://cses.fi/problemset/task/1639

// Complexity:
// O(m x n)

// How to use:
// memset(dp, -1, sizeof(dp));
// string a, b;
// edit_distance(a, b, (int)a.size(), (int)b.size());

// Notes:
// Size of dp matriz is m x n

int dp[MAX][MAX];

int edit_distance(string &str1, string &str2, int m, int n) {
    if (m == 0) return n;
    if (n == 0) return m;

    if (dp[m][n] != -1) return dp[m][n];

    if (str1[m - 1] == str2[n - 1]) return dp[m][n] = edit_distance(str1, str2, m - 1, n - 1);
    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)});
}
```

2.4 Knapsack

```
int val[MAXN], peso[MAXN], dp[MAXN][MAXS];

int knapsack(int n, int m){ // n Objetos | Peso max
    for(int i=0;i<=n;i++){
       for(int j=0;j<=m;j++){
            if(i==0 or j==0)
                dp[i][j] = 0;
            else if(peso[i-1]<=j)
                dp[i][j] = max(val[i-1]+dp[i-1][j-peso[i-1]], dp[i-1][j]);
            else
                dp[i][j] = dp[i-1][j];
       }
    }
    return dp[n][m];
}
```

2.5 Digits

```
// achar a quantidade de numeros menores que R que possuem no maximo 3 digitos nao nulos
// a ideia eh utilizar da ordem lexicografica para checar isso pois se temos por exemplo
// o numero 8500, a gente sabe que se pegarmos o numero 7... qualquer digito depois do 7
// sera necessariamente menor q 8500

string r;
int tab[20][2][5];

// i - digito de R
// menor - ja pegou um numero menor que um digito de R
// qt - quantidade de digitos nao nulos
int dp(int i, bool menor, int qt){
    if(qt > 3) return 0;
    if(i >= r.size()) return 1;
    if(tab[i][menor][qt] != -1) return tab[i][menor][qt];

    int dr = r[i]-'0';
    int res = 0;

    for(int d = 0; d <= 9; d++) {
        int dnn = qt + (d > 0);
        if(menor == true) {
            res += dp(i+1, true, dnn);
        }
        else if(d < dr) {
            res += dp(i+1, true, dnn);
        }
        else if(d == dr) {
            res += dp(i+1, false, dnn);
        }
    }

    return tab[i][menor][qt] = res;
}
```

2.6 Coins

```
int tb[1005];
int n;
vector<int> moedas;

int dp(int i){
  if(i >= n)
    return 0;
  if(tb[i] != -1)
    return tb[i];

  tb[i] = max(dp(i+1), dp(i+2) + moedas[i]);
  return tb[i];
}

int main(){
  memset(tb,-1,sizeof(tb));
}
```

2.7 Minimum Coin Change

```
int n;
vector<int> valores;
```

```

int tabela[1005];

int dp(int k){
  if(k == 0){
    return 0;
  }
  if(tabela[k] != -1)
    return tabela[k];
  int melhor = 1e9;
  for(int i = 0; i < n; i++){
    if(valores[i] <= k)
      melhor = min(melhor,1 + dp(k - valores[i]));
  }
  return tabela[k] = melhor;
}
```

2.8 Kadane

```
// achar uma subsequencia continua no array que a soma seja a maior possivel
// nesse caso vc precisa multiplicar exatamente 1 elemento da subsequencia
// e achar a maior soma com isso

int n, x, arr[MAX], tab[MAX][2]; // tab[maior resposta no intervalo][foi multiplicado ou ão]

int dp(int i, bool mult) {
    if (i == n-1) {
        if (!mult) return arr[n-1]*x;
        return arr[n-1];
    }
    if (tab[i][mult] != -1) return tab[i][mult];

    int res;

    if (mult) {
        res = max(arr[i], arr[i] + dp(i+1, 1));
    }
    else {
        res = max({
            arr[i]*x,
            arr[i]*x + dp(i+1, 1),
            arr[i] + dp(i+1, 0)
        });
    }

    return tab[i][mult] = res;
}

int main() {

    memset(tab, -1, sizeof(tab));

    int ans = -oo;
    for (int i = 0; i < n; i++) {
        ans = max(ans, dp(i, 0));
    }

    return 0;
}



int ans = a[0], ans_l = 0, ans_r = 0;
int sum = 0, minus_pos = -1;

for (int r = 0; r < n; ++r) {
    sum += a[r];
    if (sum > ans) {
        ans = sum;
        ans_l = minus_pos + 1;
        ans_r = r;
    }
    if (sum < 0) {
        sum = 0;
        minus_pos = r;
    }
}
```

3 Template

3.1 Template

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

#define int long long
#define optimize std::ios::sync_with_stdio(false); cin.tie(NULL);
#define vi vector<int>
#define ll long long
#define pb push_back
#define mp make_pair
#define ff first
#define ss second
#define pii pair<int, int>
#define MOD 1000000007
#define sqr(x) ((x) * (x))
#define all(x) (x).begin(), (x).end()
#define FOR(i, j, n) for (int i = j; i < n; i++)
#define qle(i, n) (i == n ? "\n" : " ")
#define endl "\n"
const int oo = 1e9;
const int MAX = 1e6;

int32_t main(){ optimize;

    return 0;
}
```

3.2 Template Clean

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

// Print with precision
// cout << fixed << setprecision(12) << value << endl;

// File as input and output
// freopen("input.txt", "r", stdin);
// freopen("output.txt", "w", stdout);

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

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



    return 0;
}
```

4 Strings

4.1 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;
}
```

4.2 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;
}
```

4.3 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;
}
```

4.4 Lcs

```cpp
// Description:
// Finds the longest common subsquence between two string

// Problem:
// https://codeforces.com/gym/103134/problem/B

// Complexity:
// O(mn) where m and n are the length of the strings

string lcsAlgo(string s1, string s2, int m, int n) {
  int LCS_table[m + 1][n + 1];

  for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
      if (i == 0 || j == 0)
        LCS_table[i][j] = 0;
      else if (s1[i - 1] == s2[j - 1])
        LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
      else
        LCS_table[i][j] = max(LCS_table[i - 1][j], LCS_table[i][j - 1]);
    }
  }

  int index = LCS_table[m][n];
  char lcsAlgo[index + 1];
  lcsAlgo[index] = '\0';

  int i = m, j = n;
  while (i > 0 && j > 0) {
    if (s1[i - 1] == s2[j - 1]) {
      lcsAlgo[index - 1] = s1[i - 1];
      i--;
      j--;
      index--;
    }

    else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
      i--;
    else
      j--;
  }

  return lcsAlgo;
}
```

4.5 Trie

```cpp
const int K = 26;

struct Vertex {
    int next[K];
    bool output = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].output = true;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
```

```
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

4.6 Z-function

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

5 Misc

5.1 Split

```
vector<string> split(string txt, char key = ' '){
    vector<string> ans;

    string palTemp = "";
    for(int i = 0; i < txt.size(); i++){

        if(txt[i] == key){
            if(palTemp.size() > 0){
                ans.push_back(palTemp);
                palTemp = "";
            }
        } else{
            palTemp += txt[i];
        }

    }

    if(palTemp.size() > 0)
        ans.push_back(palTemp);

    return ans;
}
```

5.2 Int128

```
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
```

6 Graphs

6.1 Centroid Find

```
// Description:
// Indexed at zero
// Find a centroid, that is a node such that when it is appointed the root of the tree,
// each subtree has at most floor(n/2) nodes.

// Problem:
// https://cses.fi/problemset/task/2079/

// Complexity:
// O(n)

// How to use:
// get_subtree_size(0);
// cout << get_centroid(0) + 1 << endl;

int n;
vector<int> adj[MAX];
int subtree_size[MAX];

int get_subtree_size(int node, int par = -1) {
  int &res = subtree_size[node];
  res = 1;
  for (int i : adj[node]) {
    if (i == par) continue;
    res += get_subtree_size(i, node);
  }
  return res;
}

int get_centroid(int node, int par = -1) {
  for (int i : adj[node]) {
    if (i == par) continue;

    if (subtree_size[i] * 2 > n) { return get_centroid(i, node); }
  }
  return node;
}

int main() {
  cin >> n;
  for (int i = 0; i < n - 1; i++) {
    int u, v; cin >> u >> v;
    u--; v--;
    adj[u].push_back(v);
    adj[v].push_back(u);
  }

  get_subtree_size(0);
  cout << get_centroid(0) + 1 << endl;
}
```

6.2 Bipartite

```cpp
const int NONE = 0, BLUE = 1, RED = 2;
vector<vector<int>> graph(100005);
vector<bool> visited(100005);
int color[100005];

bool bfs(int s = 1){

    queue<int> q;
    q.push(s);
    color[s] = BLUE;

    while (not q.empty()){
        auto u = q.front(); q.pop();

        for (auto v : graph[u]){
            if (color[v] == NONE){
                color[v] = 3 - color[u];
                q.push(v);
            }
            else if (color[v] == color[u]){
                return false;
            }
        }
    }

    return true;
}

bool is_bipartite(int n){

    for (int i = 1; i<=n; i++)
        if (color[i] == NONE and not bfs(i))
            return false;

    return true;
}
```

6.3 Prim

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

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 Ford Fulkerson Edmonds Karp

```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

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

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

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

6.5 Floyd Warshall

```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;

const int MAX  = 507;
const long long INF = 0x3f3f3f3f3f3f3f3fLL;

ll dist[MAX][MAX];
int n;

void floyd_warshall() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) dist[i][j] = 0;
            else if (!dist[i][j]) dist[i][j] = INF;
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                // trata o caso no qual o grafo tem arestas com peso negativo
                if (dist[i][k] < INF && dist[k][j] < INF){
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }
}
```

6.6 Lca

```cpp
// Description:
// Find the lowest common ancestor between two nodes in a tree

// Problem:
// https://cses.fi/problemset/task/1135

// Complexity:
// O(log n)

// How to use:
// preprocess();
// lca(a, b);

// Notes
// To calculate the distance between two nodes use the following formula
// level_peso[a] + level_peso[b] - 2*level_peso[lca(a, b)]

const int MAX = 2e5+10;
const int BITS = 30;

vector<pii> adj[MAX];
vector<bool> visited(MAX);

int up[MAX][BITS + 1];
int level[MAX];
int level_peso[MAX];

void find_level() {
  queue<pii> q;

  q.push(mp(1, 0));
  visited[1] = true;

  while (!q.empty()) {
    auto [v, depth] = q.front();
    q.pop();
    level[v] = depth;

    for (auto [u,d] : adj[v]) {
      if (!visited[u]) {
        visited[u] = true;
        up[u][0] = v;
        q.push(mp(u, depth + 1));
      }
    }
  }
}

void find_level_peso() {
  queue<pii> q;

  q.push(mp(1, 0));
  visited[1] = true;

  while (!q.empty()) {
    auto [v, depth] = q.front();
    q.pop();
    level_peso[v] = depth;

    for (auto [u,d] : adj[v]) {
      if (!visited[u]) {
        visited[u] = true;
        up[u][0] = v;
        q.push(mp(u, depth + d));
      }
    }
  }
}

int lca(int a, int b) {
   // get the nodes to the same level
    int mn = min(level[a], level[b]);

    for (int j = 0; j <= BITS; j++) {
      if (a != -1 && ((level[a] - mn) & (1 << j))) a = up[a][j];
      if (b != -1 && ((level[b] - mn) & (1 << j))) b = up[b][j];
    }

    // special case
    if (a == b) return a;

    // binary search
    for (int j = BITS; j >= 0; j--) {
      if (up[a][j] != up[b][j]) {
        a = up[a][j];
        b = up[b][j];
      }
    }
    return up[a][0];
}

void preprocess() {
  visited = vector<bool>(MAX, false);
  find_level();
  visited = vector<bool>(MAX, false);
  find_level_peso();

  for (int j = 1; j <= BITS; j++) {
    for (int i = 1; i <= n; i++) {
      if (up[i][j - 1] != -1) up[i][j] = up[up[i][j - 1]][j - 1];
    }
  }
}
```

6.7 Bellman Ford

```
struct edge
{
    int a, b, cost;
};

int n, m, v;
vector<edge> e;
const int INF = 1000000000;

void solve()
{
    vector<int> d (n, INF);
    d[v] = 0;
    for (int i=0; i<n-1; ++i)
        for (int j=0; j<m; ++j)
            if (d[e[j].a] < INF)
                d[e[j].b] = min (d[e[j].b], d[e[j].a] + e[j].cost);
}
```

6.8 Dinic

```
// 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;
}

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;
}
```

6.9 2sat

```
// Description:
```

```cpp
// 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

#include <bits/stdc++.h>
#define pb push_back
#define mp make_pair
#define pii pair<int, int>
#define ff first
#define ss second

using namespace std;

struct SAT {
    int nodes;
    int curr = 0;
    int component = 0;
    vector<vector<int>> adj;
    vector<vector<int>> rev;
    vector<vector<int>> condensed;
    vector<pii> departure;
    vector<bool> visited;
    vector<int> scc;
    vector<int> order;

    // 1 to nodes
    // nodes + 1 to 2 * nodes
    SAT(int nodes) : nodes(nodes) {
        adj.resize(2 * nodes + 1);
        rev.resize(2 * nodes + 1);
        visited.resize(2 * nodes + 1);
        scc.resize(2 * nodes + 1);
    }

    void add_imp(int a, int b) {
        adj[a].pb(b);
        rev[b].pb(a);
    }

    int get_not(int a) {
        if (a > nodes) return a - nodes;
        return a + nodes;
    }

    void add_or(int a, int b) {
        add_imp(get_not(a), b);
        add_imp(get_not(b), a);
    }

    void add_nor(int a, int b) {
        add_or(get_not(a), get_not(b));
    }

    void add_and(int a, int b) {
        add_or(get_not(a), b);
        add_or(a, get_not(b));
        add_or(a, b);
    }

    void add_nand(int a, int b) {
        add_or(get_not(a), b);
        add_or(a, get_not(b));
        add_or(get_not(a), get_not(b));
    }

    void add_xor(int a, int b) {
        add_or(a, b);
        add_or(get_not(a), get_not(b));
    }

    void add_xnor(int a, int b) {
        add_or(get_not(a), b);
        add_or(a, get_not(b));
    }

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

        for (auto u : adj[v]) {
            if (!visited[u]) departure_time(u);
        }

        departure.pb(mp(++curr, v));
    }

    void find_component(int v, int component) {
        scc[v] = component;
        visited[v] = true;

        for (auto u : rev[v]) {
            if (!visited[u]) find_component(u, component);
        }
    }

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

        for (auto u : condensed[v]) {
            if (!visited[u]) topological_order(u);
        }

        order.pb(v);
    }

    bool is_possible() {
        component = 0;
        for (int i = 1; i <= 2 * nodes; i++) {
            if (!visited[i]) departure_time(i);
        }

        sort(departure.begin(), departure.end(), greater<pii>());

        visited.assign(2 * nodes + 1, false);

        for (auto [_, node] : departure) {
            if (!visited[node]) find_component(node, ++component);
        }

        for (int i = 1; i <= nodes; i++) {
            if (scc[i] == scc[i + nodes]) return false;
        }

        return true;
    }

    int find_value(int e, vector<int> &ans) {
        if (e > nodes && ans[e - nodes] != 2) return !ans[e - nodes];
        if (e <= nodes && ans[e + nodes] != 2) return !ans[e + nodes];
        return 0;
    }

    vector<int> find_ans() {
        condensed.resize(component + 1);
```

```
        for (int i = 1; i <= 2 * nodes; i++) {
            for (auto u : adj[i]) {
                if (scc[i] != scc[u]) condensed[scc[i]].pb(scc[u]);
            }
        }

        visited.assign(component + 1, false);

        for (int i = 1; i <= component; i++) {
            if (!visited[i]) topological_order(i);
        }

        reverse(order.begin(), order.end());

        // 0 - false
        // 1 - true
        // 2 - no value yet
        vector<int> ans(2 * nodes + 1, 2);

        vector<vector<int>> belong(component + 1);

        for (int i = 1; i <= 2 * nodes; i++) {
            belong[scc[i]].pb(i);
        }

        for (auto p : order) {
            for (auto e : belong[p]) {
                ans[e] = find_value(e, ans);
            }
        }

        return ans;
    }
};

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

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

    SAT sat = SAT(m);

    for (int i = 0; i < n; i++) {
        char op1, op2; int a, b; cin >> op1 >> a >> op2 >> b;
        if (op1 == '+' && op2 == '+') sat.add_or(a, b);
        if (op1 == '-' && op2 == '-') sat.add_or(sat.get_not(a), sat.get_not(b));
        if (op1 == '+' && op2 == '-') sat.add_or(a, sat.get_not(b));
        if (op1 == '-' && op2 == '+') sat.add_or(sat.get_not(a), b);
    }

    if (!sat.is_possible()) cout << "IMPOSSIBLE\n";
    else {
        vector<int> ans = sat.find_ans();
        for (int i = 1; i <= m; i++) {
            cout << (ans[i] == 1 ? '+' : '-') << ' ';
        }
        cout << '\n';
    }

    return 0;
}
```

6.10 Find Cycle

```
bitset<MAX> visited;
vector<int> path;
vector<int> adj[MAX];

bool dfs(int u, int p){

    if (visited[u]) return false;

    path.pb(u);
    visited[u] = true;

    for (auto v : adj[u]){
        if (visited[v] and u != v and p != v){
            path.pb(v); return true;
        }

        if (dfs(v, u)) return true;
    }

    path.pop_back();
    return false;
}

bool has_cycle(int N){

    visited.reset();

    for (int u = 1; u <= N; ++u){
        path.clear();
        if (not visited[u] and dfs(u,-1))
            return true;

    }

    return false;
}
```

6.11 Cycle Path Recovery

```
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
```

```
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

6.12 Centroid Decomposition

```
int n;
vector<set<int>> adj;
vector<char> ans;

vector<bool> removed;

vector<int> subtree_size;

int dfs(int u, int p = 0) {
  subtree_size[u] = 1;

  for(int v : adj[u]) {
    if(v != p && !removed[v]) {
      subtree_size[u] += dfs(v, u);
        }
    }

  return subtree_size[u];
}

int get_centroid(int u, int sz, int p = 0) {
  for(int v : adj[u]) {
    if(v != p && !removed[v]) {
      if(subtree_size[v]*2 > sz) {
        return get_centroid(v, sz, u);
            }
        }
    }

  return u;
}

char get_next(char c) {
    if (c != 'Z') return c + 1;
    return '$';
}

bool flag = true;

void solve(int node, char c) {
  int center = get_centroid(node, dfs(node));
    ans[center] = c;
    removed[center] = true;

    for (auto u : adj[center]) {
        if (!removed[u]) {
            char next = get_next(c);
            if (next == '$') {
                flag = false;
                return;
            }
            solve(u, next);
        }
    }
}

int32_t main(){
    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;
}
```

6.13 Tarjan Bridge

```
// 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);
    }
}
```

6.14 Small To Large

```
// Problem:
// https://codeforces.com/contest/600/problem/E

void process_colors(int curr, int parent) {

  for (int n : adj[curr]) {
    if (n != parent) {
      process_colors(n, curr);

            if (colors[curr].size() < colors[n].size()) {
                sum_num[curr] = sum_num[n];
                vmax[curr] = vmax[n];
        swap(colors[curr], colors[n]);
      }

      for (auto [item,vzs] : colors[n]) {
                if(colors[curr][item]+vzs > vmax[curr]){
                    vmax[curr] = colors[curr][item] + vzs;
                    sum_num[curr] = item;
                }
                else if(colors[curr][item]+vzs == vmax[curr]){
                    sum_num[curr] += item;
                }

                colors[curr][item] += vzs;
      }
    }
  }

}


int32_t main() {

  int n; cin >> n;

  for (int i = 1; i <= n; i++) {
    int a; cin >> a;
    colors[i][a] = 1;
        vmax[i] = 1;
        sum_num[i] = a;
  }

  for (int i = 1; i < n; i++) {
    int a, b; cin >> a >> b;

    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  process_colors(1, 0);

  for (int i = 1; i <= n; i++) {
    cout << sum_num[i] << (i < n ? " " : "\n");
  }

    return 0;

}

```

6.15 Tree Diameter

```
#include<bits/stdc++.h>

using namespace std;

const int MAX = 3e5+17;

vector<int> adj[MAX];
bool visited[MAX];

int max_depth = 0, max_node = 1;

void dfs (int v, int depth) {
    visited[v] = true;

    if (depth > max_depth) {
        max_depth = depth;
        max_node = v;
    }

    for (auto u : adj[v]) {
        if (!visited[u]) dfs(u, depth + 1);
    }
}

int tree_diameter() {
    dfs(1, 0);
    max_depth = 0;
    for (int i = 0; i < MAX; i++) visited[i] = false;
    dfs(max_node, 0);
    return max_depth;
}
```

6.16 Dijkstra

```
const int MAX = 2e5+7;
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj(MAX);

void dijkstra(int s, vector<int> & d, vector<int> & p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);

    d[s] = 0;
    set<pair<int, int>> q;
    q.insert({0, s});
    while (!q.empty()) {
        int v = q.begin()->second;
        q.erase(q.begin());

        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                q.erase({d[to], to});
                d[to] = d[v] + len;
                p[to] = v;
                q.insert({d[to], to});
            }
        }
    }
}

vector<int> restore_path(int s, int t) {
    vector<int> path;

    for (int v = t; v != s; v = p[v])
        path.push_back(v);
```

```
    path.push_back(s);

    reverse(path.begin(), path.end());
    return path;
}

int adj[MAX][MAX];
int dist[MAX];
int minDistance(int dist[], bool sptSet[], int V) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

void dijkstra(int src, int V) {

    bool sptSet[V];
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet, V);

        sptSet[u] = true;


        for (int v = 0; v < V; v++)
            if (!sptSet[v] && adj[u][v]
                && dist[u] != INT_MAX
                && dist[u] + adj[u][v] < dist[v])
                dist[v] = dist[u] + adj[u][v];
    }
}
```

6.17 Kruskall

```
struct DSU {
    int n;
    vector<int> link, sizes;

    DSU(int n) {
        this->n = n;
        link.assign(n+1, 0);
        sizes.assign(n+1, 1);

        for (int i = 0; i <= n; i++)
            link[i] = i;
    }

    int find(int x) {
        while (x != link[x])
            x = link[x];

        return x;
    }

    bool same(int a, int b) {
        return find(a) == find(b);
    }

    void unite(int a, int b) {
        a = find(a);
        b = find(b);

        if (a == b) return;

        if (sizes[a] < sizes[b])
            swap(a, b);

        sizes[a] += sizes[b];
        link[b] = a;
    }
};

struct Edge {
    int u, v;
    long long weight;

    Edge() {}

    Edge(int u, int v, long long weight) : u(u), v(v), weight(weight) {}

    bool operator<(const Edge& other) const {
        return weight < other.weight;
    }

    bool operator>(const Edge& other) const {
        return weight > other.weight;
    }
};

vector<Edge> kruskal(vector<Edge> edges, int n) {
    vector<Edge> result; // arestas da MST
    long long cost = 0;

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

    DSU dsu(n);

    for (auto e : edges) {
        if (!dsu.same(e.u, e.v)) {
            cost += e.weight;
            result.push_back(e);
            dsu.unite(e.u, e.v);
        }
    }

    return result;
}
```

7 Geometry

7.1 2d

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

// typedef ll cod;
// bool eq(cod a, cod b){ return (a==b); }
typedef ld cod;
bool eq(cod a, cod b){ return abs(a - b) <= EPS; }

struct point{
    cod x, y;
    int id;
    point(cod x=0, cod y=0): x(x), y(y){}

    point operator+(const point &o) const{ return {x+o.x, y+o.y}; }
    point operator-(const point &o) const{ return {x-o.x, y-o.y}; }
    point operator*(cod t) const{ return {x*t, y*t}; }
    point operator/(cod t) const{ return {x/t, y/t}; }
```

```
    cod operator*(const point &o) const{ return x * o.x + y * o.y; }
    cod operator^(const point &o) const{ return x * o.y - y * o.x; }
    bool operator<(const point &o) const{
        return (eq(x, o.x) ? y < o.y : x < o.x);
    }
    bool operator==(const point &o) const{
        return eq(x, o.x) and eq(y, o.y);
    }
  friend ostream& operator<<(ostream& os, point p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};

int ccw(point a, point b, point e){ // -1=dir; 0=collinear; 1=esq;
    cod tmp = (b-a) ^ (e-a); // vector from a to b
    return (tmp > EPS) - (tmp < -EPS);
}

ld norm(point a){ // Modulo
    return sqrt(a * a);
}
cod norm2(point a){
    return a * a;
}
bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0));
}
point rotccw(point p, ld a){
    // a = PI*a/180; // graus
    return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+p.x*sin(a)));
}
point rot90cw(point a) { return point(a.y, -a.x); };
point rot90ccw(point a) { return point(-a.y, a.x); };

ld proj(point a, point b){ // a sobre b
    return a*b/norm(b);
}
ld angle(point a, point b){ // em radianos
    ld ang = a*b / norm(a) / norm(b);
    return acos(max(min(ang, (ld)1), (ld)-1));
}
ld angle_vec(point v){
    // return 180/PI*atan2(v.x, v.y); // graus
    return atan2(v.x, v.y);
}
ld order_angle(point a, point b){ // from a to b ccw (a in front of b)
    ld aux = angle(a,b)*180/PI;
    return ((a^b)<=0 ? aux:360-aux);
}
bool angle_less(point a1, point b1, point a2, point b2){ // ang(a1,b1) <= ang(a2,b2)
    point p1((a1*b1), abs((a1^b1)));
    point p2((a2*b2), abs((a2^b2)));
    return (p1^p2) <= 0;
}

ld area(vp &p){ // (points sorted)
    ld ret = 0;
    for(int i=2;i<(int)p.size();i++)
        ret += (p[i]-p[0])^(p[i-1]-p[0]);
    return abs(ret/2);
}
ld areaT(point &a, point &b, point &c){
    return abs((b-a)^(c-a))/2.0;
}

point center(vp &A){
    point c = point();
    int len = A.size();
    for(int i=0;i<len;i++)
        c=c+A[i];
    return c/len;
}

point forca_mod(point p, ld m){
    ld cm = norm(p);
    if(cm<EPS) return point();
    return point(p.x*m/cm,p.y*m/cm);
}

ld param(point a, point b, point v){
    // v = t*(b-a) + a // return t;
    // assert(line(a, b).inside_seg(v));
    return ((v-a) * (b-a)) / ((b-a) * (b-a));
}

bool simetric(vp &a){ //ordered
    int n = a.size();
    point c = center(a);
    if(n&1) return false;
    for(int i=0;i<n/2;i++)
        if(ccw(a[i], a[i+n/2], c) != 0)
            return false;
    return true;
}

point mirror(point m1, point m2, point p){
    // mirror point p around segment m1m2
    point seg = m2-m1;
    ld t0 = ((p-m1)*seg) / (seg*seg);
    point ort = m1 + seg*t0;
    point pm = ort-(p-ort);
    return pm;
}


////////////
//  Line  //
////////////

struct line{
    point p1, p2;
    cod a, b, c; // ax+by+c = 0;
    // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
    line(point p1=0, point p2=0): p1(p1), p2(p2){
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = p1 ^ p2;
    }
    line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
    {
        // Gera os pontos p1 p2 dados os coeficientes
        // isso aqui eh um lixo mas quebra um galho kkkkkk
        if(b==0){
            p1 = point(1, -c/a);
            p2 = point(0, -c/a);
        }else{
            p1 = point(1, (-c-a*1)/b);
            p2 = point(0, -c/b);
        }
    }

    cod eval(point p){
        return a*p.x+b*p.y+c;
    }
    bool inside(point p){
        return eq(eval(p), 0);
    }
    point normal(){
        return point(a, b);
    }
```

```
    bool inside_seg(point p){
        return (
            ((p1-p) ^ (p2-p)) == 0 and
            ((p1-p) * (p2-p)) <= 0
        );
    }

};

// be careful with precision error
vp inter_line(line l1, line l2){
    ld det = l1.a*l2.b - l1.b*l2.a;
    if(det==0) return {};
    ld x = (l1.b*l2.c - l1.c*l2.b)/det;
    ld y = (l1.c*l2.a - l1.a*l2.c)/det;
    return {point(x, y)};
}

// segments not collinear
vp inter_seg(line l1, line l2){
    vp ans = inter_line(l1, l2);
    if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.inside_seg(ans[0]))
        return {};
    return ans;
}
bool seg_has_inter(line l1, line l2){
    return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.p2, l2.p2) < 0 and
           ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.p2, l1.p2) < 0;
}

ld dist_seg(point p, point a, point b){ // point - seg
    if((p-a)*(b-a) < EPS) return norm(p-a);
    if((p-b)*(a-b) < EPS) return norm(p-b);
    return abs((p-a)^(b-a)) / norm(b-a);
}

ld dist_line(point p, line l){ // point - line
    return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
}

line bisector(point a, point b){
    point d = (b-a)*2;
    return line(d.x, d.y, a*a - b*b);
}

line perpendicular(line l, point p){ // passes through p
    return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
}


////////////
// Circle //
////////////

struct circle{
    point c; cod r;
    circle() : c(0, 0), r(0){}
    circle(const point o) : c(o), r(0){}
    circle(const point a, const point b){
        c = (a+b)/2;
        r = norm(a-c);
    }
    circle(const point a, const point b, const point cc){
        assert(ccw(a, b, cc) != 0);
        c = inter_line(bisector(a, b), bisector(b, cc))[0];
        r = norm(a-c);
    }
    bool inside(const point &a) const{
        return norm(a - c) <= r + EPS;
    }
};

pair<point, point> tangent_points(circle cr, point p){
    ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
    point p1 = rotccw(cr.c-p, -theta);
    point p2 = rotccw(cr.c-p, theta);
    assert(d1 >= cr.r);
    p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
    p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
    return {p1, p2};
}


circle incircle(point p1, point p2, point p3){
    ld m1 = norm(p2-p3);
    ld m2 = norm(p1-p3);
    ld m3 = norm(p1-p2);
    point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
    ld s = 0.5*(m1+m2+m3);
    ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
    return circle(c, r);
}

circle circumcircle(point a, point b, point c) {
    circle ans;
    point u = point((b-a).y, -(b-a).x);
    point v = point((c-a).y, -(c-a).x);
    point n = (c-b)*0.5;
    ld t = (u^n)/(v^u);
    ans.c = ((a+c)*0.5) + (v*t);
    ans.r = norm(ans.c-a);
    return ans;
}

vp inter_circle_line(circle C, line L){
    point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.p1)*(ab) / (ab*ab));
    ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s / (ab*ab);
    if (h2 < -EPS) return {};
    if (eq(h2, 0)) return {p};
    point h = (ab/norm(ab)) * sqrt(h2);
    return {p - h, p + h};
}

vp inter_circle(circle C1, circle C2){
    if(C1.c == C2.c) { assert(C1.r != C2.r); return {}; }
    point vec = C2.c - C1.c;
    ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r;
    ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 = C1.r*C1.r - p*p*d2;
    if (sum*sum < d2 or dif*dif > d2) return {};
    point mid = C1.c + vec*p, per = point(-vec.y, vec.x) * sqrt(max((ld)0, h2) / d2);
    if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
    return {mid + per, mid - per};
}

// minimum circle cover O(n) amortizado
circle min_circle_cover(vp v){
    random_shuffle(v.begin(), v.end());
    circle ans;
    int n = v.size();
    for(int i=0;i<n;i++) if(!ans.inside(v[i])){
        ans = circle(v[i]);
```

```
        for(int j=0;j<i;j++) if(!ans.inside(v[j])){
            ans = circle(v[i], v[j]);
            for(int k=0;k<j;k++) if(!ans.inside(v[k])){
                ans = circle(v[i], v[j], v[k]);
            }
        }
    }
    return ans;
}
```

8 Algorithms

8.1 Lis

```
int lis(vector<int> const& a) {
    int n = a.size();
    vector<int> d(n, 1);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i])
                d[i] = max(d[i], d[j] + 1);
        }
    }

    int ans = d[0];
    for (int i = 1; i < n; i++) {
        ans = max(ans, d[i]);
    }
    return ans;
}
```

8.2 Delta-encoding

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

int main(){
    int n, q;
    cin >> n >> q;
    int [n];
    int delta[n+2];

    while(q--){
        int l, r, x;
        cin >> l >> r >> x;
        delta[l] += x;
        delta[r+1] -= x;
    }

    int curr = 0;
    for(int i=0; i < n; i++){
        curr += delta[i];
        v[i] = curr;
    }

    for(int i=0; i< n; i++){
        cout << v[i] << ' ';
    }
    cout << '\n';

    return 0;
}
```

8.3 Subsets

```
void subsets(vector<int>& nums){
  int n = nums.size();
  int powSize = 1 << n;

  for(int counter = 0; counter < powSize; counter++){
    for(int j = 0; j < n; j++){
      if((counter & (1LL << j)) != 0) {
        cout << nums[j] << ' ';
      }
    }
    cout << '\n';
  }
}
```

8.4 Binary Search Last True

```
int last_true(int lo, int hi, function<bool(int)> f)
    {
  lo--;
  while (lo < hi) {
    int mid = lo + (hi - lo + 1) / 2;
    if (f(mid)) {
      lo = mid;
    } else {
      hi = mid - 1;
    }
  }
  return lo;
}
```

8.5 Ternary Search

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

8.6 Binary Search First True

```
int first_true(int lo, int hi, function<bool(int)> f)
     {
  hi++;
  while (lo < hi) {
    int mid = lo + (hi - lo) / 2;
    if (f(mid)) {
      hi = mid;
    } else {
      lo = mid + 1;
    }
  }
  return lo;
}
```

8.7 Biggest K

```
// Description: Gets sum of k biggest or k smallest
    elements in an array

// Problem: https://atcoder.jp/contests/abc306/tasks/
    abc306_e

// Complexity: O(log n)

struct SetSum {
```

```
    ll s = 0;
    multiset<ll> mt;
    void add(ll x){
        mt.insert(x);
        s += x;
    }
    int pop(ll x){
        auto f = mt.find(x);
        if(f == mt.end()) return 0;
        mt.erase(f);
        s -= x;
        return 1;
    }
};

struct BigK {
    int k;
    SetSum gt, mt;
    BigK(int _k){
        k = _k;
    }
    void balancear(){
        while((int)gt.mt.size() < k && (int)mt.mt.size()){
            auto p = (prev(mt.mt.end()));
            gt.add(*p);
            mt.pop(*p);
        }
        while((int)mt.mt.size() && (int)gt.mt.size() &&
        *(gt.mt.begin()) < *(prev(mt.mt.end())) ){
            ll u = *(gt.mt.begin());
            ll v = *(prev(mt.mt.end()));
            gt.pop(u); mt.pop(v);
            gt.add(v); mt.add(u);
        }
    }
    void add(ll x){
        mt.add(x);
        balancear();
    }
    void rem(ll x){
        //x = -x;
        if(mt.pop(x) == 0)
            gt.pop(x);
        balancear();
    }
};

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

    int n, k, q; cin >> n >> k >> q;

    BigK big = BigK(k);

    int arr[n] = {};

    while (q--) {
        int pos, num; cin >> pos >> num;
        pos--;
        big.rem(arr[pos]);
        arr[pos] = num;
        big.add(arr[pos]);

        cout << big.gt.s << '\n';
    }

    return 0;
}
```

9 Data Structures

9.1 Ordered Set

```
// Description:
// insert(k) - add element k to the ordered set
// erase(k) - remove element k from the ordered set
// erase(it) - remove element it points to from the ordered set
// order_of_key(k) - returns number of elements strictly smaller than k
// find_by_order(n) - return an iterator pointing to the k-th element in the ordered set (counting from zero).

// Problem:
// https://cses.fi/problemset/task/2169/

// Complexity:
// O(log n) for all operations

// How to use:
// ordered_set<int> os;
// cout << os.order_of_key(1) << '\n;
// cout << os.find_by_order(1) << '\n;

// Notes
// The ordered set only contains different elements
// By using less_equal<T> instead of less<T> on using ordered_set declaration
// The ordered_set becomes an ordered_multiset
// So the set can contain elements that are equal

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T,null_type,less<T>, rb_tree_tag,tree_order_statistics_node_update>;

void Erase(ordered_set<int>& a, int x){
    int r = a.order_of_key(x);
    auto it = a.find_by_order(r);
    a.erase(it);
}
```

9.2 Priority Queue

```
// Description:
// Keeps the largest (by default) element at the top of the queue

// Problem:
// https://cses.fi/problemset/task/1164/

// Complexity:
// O(log n) for push and pop
// O (1) for looking at the element at the top

// How to use:
// prioriy_queue<int> pq;
// pq.push(1);
// pq.top();
// pq.pop()

// Notes
// To use the priority queue keeping the smallest element at the top

priority_queue<int, vector<int>, greater<int>> pq;
```

9.3 Dsu

```cpp
#include <bits/stdc++.h>

using namespace std;

const int MAX = 1e6+17;

struct DSU {
    int n;
    vector<int> link, sizes;

    DSU(int n) {
        this->n = n;
        link.assign(n+1, 0);
        sizes.assign(n+1, 1);

        for (int i = 0; i <= n; i++)
            link[i] = i;
    }

    int find(int x) {
        while (x != link[x])
            x = link[x];

        return x;
    }

    bool same(int a, int b) {
        return find(a) == find(b);
    }

    void unite(int a, int b) {
        a = find(a);
        b = find(b);

        if (a == b) return;

        if (sizes[a] < sizes[b])
            swap(a, b);

        sizes[a] += sizes[b];
        link[b] = a;
    }

    int size(int x) {
        return sizes[x];
    }
};

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

    int cities, roads; cin >> cities >> roads;
    vector<int> final_roads;
    int ans = 0;
    DSU dsu = DSU(cities);
    for (int i = 0, a, b; i < roads; i++) {
        cin >> a >> b;
        dsu.unite(a, b);
    }

    for (int i = 2; i <= cities; i++) {
        if (!dsu.same(1, i)) {
            ans++;
            final_roads.push_back(i);
            dsu.unite(1,i);
        }
    }

    cout << ans << '\n';
    for (auto e : final_roads) {
        cout << "1 " << e << '\n';
    }

}
```

9.4 Two Sets

```cpp
// Description
// THe values are divided in two multisets so that one of them contain all values that are
// smaller than the median and the other one contains all values that are greater or equal to the median.

// Problem:
// https://atcoder.jp/contests/abc306/tasks/abc306_e
// Problem I - Maratona Feminina de çãPrograma da Unicamp 2023
// https://codeforces.com/group/WYIydkiPyE/contest/450037/attachments

// Complexity:
// Add and remove elements - O(log n)
// Return sum of biggest or smallest set or return the median - O(1)

using ll = long long;

struct TwoSets {
  multiset<int> small;
  multiset<int> big;
  ll sums = 0;
  ll sumb = 0;
  int n = 0;

  int size_small() {
    return small.size();
  }

  int size_big() {
    return big.size();
  }

  void balance() {
    while (size_small() > n / 2) {
      int v = *small.rbegin();
      small.erase(prev(small.end()));
      big.insert(v);
      sums -= v;
      sumb += v;
    }
    while (size_big() > n - n / 2) {
      int v = *big.begin();
      big.erase(big.begin());
      small.insert(v);
      sumb -= v;
      sums += v;
    }
  }

  void add(int x) {
    n++;
    small.insert(x);
    sums += x;
    while (!small.empty() && *small.rbegin() > *big.begin()) {
      int v = *small.rbegin();
      small.erase(prev(small.end()));
      big.insert(v);
      sums -= v;
      sumb += v;
    }
    balance();
```

```
  }

  bool rem(int x) {
    n--;
    auto it1 = small.find(x);
    auto it2 = big.find(x);
    bool flag = false;
    if (it1 != small.end()) {
      sums -= *it1;
      small.erase(it1);
      flag = true;
    } else if (it2 != big.end()) {
      sumb -= *it2;
      big.erase(it2);
      flag = true;
    }
    balance();
    return flag;
  }

  ll sum_small() {
    return sums;
  }

  ll sum_big() {
    return sumb;
  }

  int median() {
    return *big.begin();
  }
};
```

9.5 Dynamic Implicit Sparse

```
// Description:
// Indexed at one

// When the indexes of the nodes are too big to be stored in an array
// and the queries need to be answered online so we can't sort the nodes and compress them
// we create nodes only when they are needed so there'll be (Q*log(MAX)) nodes
// where Q is the number of queries and MAX is the maximum index a node can assume

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

// Problem:
// https://cses.fi/problemset/task/1648

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

// How to use:
// MAX is the maximum index a node can assume

// Segtree seg = Segtree(MAX);

typedef long long ftype;

const int MAX = 1e9+17;

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

    Segtree(int n) {
        this->n = n;
        create();
        create();
    }

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

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], 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 == fim) {
            seg[pos] = val;

            return;
        }

        int m = (ini + fim) >> 1;

        if (id <= m) {
            if (e[pos] == 0) e[pos] = create();
            update(e[pos], ini, m, id, val);
        } else {
            if (d[pos] == 0) d[pos] = create();
            update(d[pos], m + 1, fim, id, val);
        }

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

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

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

9.6 Segtree2d

```
// Description:
// Indexed at zero
// Given a N x M grid, where i represents the row and j the column, perform the following operations
// update(j, i) - update the value of grid[i][j]
// query(j1, j2, i1, i2) - return the sum of values inside the rectangle
// defined by grid[i1][j1] and grid[i2][j2] inclusive

// Problem:
// https://cses.fi/problemset/task/1739/
```

```cpp

// Complexity:
// Time complexity:
// O(log N * log M) for both query and update
// O(N * M) for build
// Memory complexity:
// 4 * M * N

// How to use:
// Segtree2D seg = Segtree2D(n, n);
// vector<vector<int>> v(n, vector<int>(n));
// seg.build(v);

// Notes
// Indexed at zero

struct Segtree2D {
    const int MAXN = 1025;
    int N, M;

    vector<vector<int>> seg;

    Segtree2D(int N, int M) {
        this->N = N;
        this->M = M;
        seg.resize(2*MAXN, vector<int>(2*MAXN));
    }

    void buildY(int noX, int lX, int rX, int noY, int lY, int rY, vector<vector<int>> &v){
        if(lY == rY){
            if(lX == rX){
                seg[noX][noY] = v[rX][rY];
            }else{
                seg[noX][noY] = seg[2*noX+1][noY] + seg[2*noX+2][noY];
            }
        }else{
            int m = (lY+rY)/2;

            buildY(noX, lX, rX, 2*noY+1, lY, m, v);
            buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);

            seg[noX][noY] = seg[noX][2*noY+1] + seg[noX][2*noY+2];
        }
    }

    void buildX(int noX, int lX, int rX, vector<vector<int>> &v){
        if(lX != rX){
            int m = (lX+rX)/2;

            buildX(2*noX+1, lX, m, v);
            buildX(2*noX+2, m+1, rX, v);
        }

        buildY(noX, lX, rX, 0, 0, M - 1, v);
    }

    void updateY(int noX, int lX, int rX, int noY, int lY, int rY, int y){
        if(lY == rY){
            if(lX == rX){
                seg[noX][noY] = !seg[noX][noY];
            }else{
                seg[noX][noY] = seg[2*noX+1][noY] + seg[2*noX+2][noY];
            }
        }else{
            int m = (lY+rY)/2;

            if(y <= m){
                updateY(noX, lX, rX, 2*noY+1,lY, m, y);
            }else if(m < y){
                updateY(noX, lX, rX, 2*noY+2, m+1, rY, y);
            }

            seg[noX][noY] = seg[noX][2*noY+1] + seg[noX][2*noY+2];
        }
    }

    void updateX(int noX, int lX, int rX, int x, int y){
        int m = (lX+rX)/2;

        if(lX != rX){
            if(x <= m){
                updateX(2*noX+1, lX, m, x, y);
            }else if(m < x){
                updateX(2*noX+2, m+1, rX, x, y);
            }
        }

        updateY(noX, lX, rX, 0, 0, M - 1, y);
    }

    int queryY(int noX, int noY, int lY, int rY, int aY, int bY){
        if(aY <= lY && rY <= bY) return seg[noX][noY];

        int m = (lY+rY)/2;

        if(bY <= m) return queryY(noX, 2*noY+1, lY, m, aY, bY);
        if(m < aY) return queryY(noX, 2*noY+2, m+1, rY, aY, bY);

        return queryY(noX, 2*noY+1, lY, m, aY, bY) + queryY(noX, 2*noY+2, m+1, rY, aY, bY);
    }

    int queryX(int noX, int lX, int rX, int aX, int bX, int aY, int bY){
        if(aX <= lX && rX <= bX) return queryY(noX, 0, 0, M - 1, aY, bY);

        int m = (lX+rX)/2;

        if(bX <= m) return queryX(2*noX+1, lX, m, aX, bX, aY, bY);
        if(m < aX) return queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);

        return queryX(2*noX+1, lX, m, aX, bX, aY, bY) + queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
    }

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

    int query(int aX, int bX, int aY, int bY) {
        return queryX(0, 0, N - 1, aX, bX, aY, bY);
    }

    void update(int x, int y) {
        updateX(0, 0, N - 1, x, y);
    }
};
```

9.7 Minimum And Amount

```cpp
// Description:
// Query - get minimum element in a range (l, r) inclusive
// and also the number of times it appears in that range
// Update - update element at position id to a value val

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

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

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

#define pii pair<int, int>
#define mp make_pair
#define ff first
#define ss second

const int INF = 1e9+17;

typedef pii ftype;

struct Segtree {
    vector<ftype> seg;
    int n;
    const ftype NEUTRAL = mp(INF, 0);

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

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

    ftype f(ftype a, ftype b) {
        if (a.ff < b.ff) return a;
        if (b.ff < a.ff) return b;

        return mp(a.ff, a.ss + b.ss);
    }

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

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

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

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

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

        if (ini == id && fim == id) {
            seg[pos] = mp(val, 1);

            return;
        }

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

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

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

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            if (ini < (int)v.size()) {
                seg[pos] = mp(v[ini], 1);
            }
            return;
        }

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

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

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

    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.ff << ' ' << e.ss << '\n';
        }
        cout << '\n';
    }
};
```

9.8 Lazy Addition To Segment

```cpp
// Description:
// Query - get sum of elements from range (l, r) inclusive
// Update - add a value val to elementos from range (l, r) inclusive

// Problem:
// https://codeforces.com/edu/course/2/lesson/5/1/practice/contest/279634/problem/A

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

// How to use:
// Segtree seg = Segtree(n);
```

```cpp
// seg.build(v);

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

const long long INF = 1e18+10;

typedef long long ftype;

struct Segtree {
    vector<ftype> seg;
    vector<ftype> lazy;
    int n;
    const ftype NEUTRAL = 0;
    const ftype NEUTRAL_LAZY = -1; // change to -INF if there are negative numbers

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

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

    ftype apply_lazy(ftype a, ftype b, int len) {
        if (b == NEUTRAL_LAZY) return a;
        if (a == NEUTRAL_LAZY) return b * len;
        else return a + b * len;
    }

    void propagate(int pos, int ini, int fim) {
        if (ini == fim) {
            return;
        }

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

        lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
        lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);

        seg[e] = apply_lazy(seg[e], lazy[pos], m - ini + 1);
        seg[d] = apply_lazy(seg[d], lazy[pos], fim - m);

        lazy[pos] = NEUTRAL_LAZY;
    }

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

    ftype query(int pos, int ini, int fim, int p, int q) {
        propagate(pos, ini, fim);

        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 p, int q, int val) {
        propagate(pos, ini, fim);

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

        if (ini >= p && fim <= q) {
            lazy[pos] = apply_lazy(lazy[pos], val, 1);
            seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);

            return;
        }

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

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

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

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            if (ini < (int)v.size()) {
                seg[pos] = 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 p, int q, int val) {
        update(0, 0, n - 1, p, q, val);
    }

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

    void debug() {
        for (auto e : seg) {
            cout << e << ' ';
        }
        cout << '\n';
        for (auto e : lazy) {
            cout << e << ' ';
        }
        cout << '\n';
        cout << '\n';
    }
```

```
};
```

9.9 Segment With Maximum Sum

```
// Description:
// Query - get sum of segment that is maximum among all segments
// E.g
// Array: 5 -4 4 3 -5
// Maximum segment sum: 8 because 5 + (-4) + 4 + 3 = 8
// Update - update element at position id to a value val

// Problem:
// https://codeforces.com/edu/course/2/lesson/4/2/practice/contest/273278/problem/A

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

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

// Notes
// The maximum segment sum can be a negative number
// In that case, taking zero elements is the best choice
// So we need to take the maximum between 0 and the query
// max(0LL, seg.query(0, n).max_seg)

using ll = long long;

typedef ll ftype_node;

struct Node {
    ftype_node max_seg;
    ftype_node pref;
    ftype_node suf;
    ftype_node sum;

    Node(ftype_node max_seg, ftype_node pref, ftype_node suf, ftype_node sum) : max_seg(max_seg), pref(pref), suf(suf), sum(sum) {};
};

typedef Node ftype;

struct Segtree {
    vector<ftype> seg;
    int n;
    const ftype NEUTRAL = Node(0, 0, 0, 0);

    Segtree(int n) {
        int sz = 1;
        // potencia de dois mais proxima
        while (sz < n) sz *= 2;
        this->n = sz;

        // numero de nos da seg
        seg.assign(2*sz, NEUTRAL);
    }

    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';
    }
};
```

9.10 Range Query Point Update

```
// 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;
    }

    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] = 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) {
            if (ini < (int)v.size()) {
                seg[pos] = 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 << ' ';
        }
        cout << '\n';
    }
};
```

9.11 Lazy Assignment To Segment

```
const long long INF = 1e18+10;

typedef long long ftype;
```

```
struct Segtree {
    vector<ftype> seg;
    vector<ftype> lazy;
    int n;
    const ftype NEUTRAL = 0;
    const ftype NEUTRAL_LAZY = -1; // Change to -INF if there are negative numbers

    Segtree(int n) {
        int sz = 1;
        // potencia de dois mais proxima
        while (sz < n) sz *= 2;
        this->n = sz;

        // numero de nos da seg
        seg.assign(2*sz, NEUTRAL);
        lazy.assign(2*sz, NEUTRAL_LAZY);
    }

    ftype apply_lazy(ftype a, ftype b, int len) {
        if (b == NEUTRAL_LAZY) return a;
        if (a == NEUTRAL_LAZY) return b * len;
        else return b * len;
    }

    void propagate(int pos, int ini, int fim) {
        if (ini == fim) {
            return;
        }

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

        lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
        lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);

        seg[e] = apply_lazy(seg[e], lazy[pos], m - ini + 1);
        seg[d] = apply_lazy(seg[d], lazy[pos], fim - m);

        lazy[pos] = NEUTRAL_LAZY;
    }

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

    ftype query(int pos, int ini, int fim, int p, int q) {
        propagate(pos, ini, fim);

        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 p, int q, int val) {
        propagate(pos, ini, fim);

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

        if (ini >= p && fim <= q) {
            lazy[pos] = apply_lazy(lazy[pos], val, 1);
            seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);

            return;
        }

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

        update(e, ini, m, p, q, val);
        update(d, m + 1, fim, p, q, 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] = 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 p, int q, int val) {
        update(0, 0, n - 1, p, q, val);
    }

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

    void debug() {
        for (auto e : seg) {
            cout << e << ' ';
        }
        cout << '\n';
        for (auto e : lazy) {
            cout << e << ' ';
        }
        cout << '\n';
        cout << '\n';
    }
};
```

9.12 Lazy Dynamic Implicit Sparse

```
// Description:
```

```
// Indexed at one

// When the indexes of the nodes are too big to be stored in an array
// and the queries need to be answered online so we can't sort the nodes and compress them
// we create nodes only when they are needed so there'll be (Q*log(MAX)) nodes
// where Q is the number of queries and MAX is the maximum index a node can assume

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

// Problem:
// https://oj.uz/problem/view/IZhO12_apple

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

// How to use:
// MAX is the maximum index a node can assume
// Create a default null node
// Create a node to be the root of the segtree

// Segtree seg = Segtree(MAX);

const int MAX = 1e9+10;
const long long INF = 1e18+10;

typedef long long ftype;

struct Segtree {
    vector<ftype> seg, d, e, lazy;
    const ftype NEUTRAL = 0;
    const ftype NEUTRAL_LAZY = -1; // change to -INF if the elements can be negative
    int n;

    Segtree(int n) {
        this->n = n;
        create();
        create();
    }

    ftype apply_lazy(ftype a, ftype b, int len) {
        if (b == NEUTRAL_LAZY) return a;
        else return b * len; // change to a + b * len to add to an element instead of updating it
    }

    void propagate(int pos, int ini, int fim) {
        if (seg[pos] == 0) return;

        if (ini == fim) {
            return;
        }

        int m = (ini + fim) >> 1;

        if (e[pos] == 0) e[pos] = create();
        if (d[pos] == 0) d[pos] = create();

        lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[pos], 1);
        lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[pos], 1);

        seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[pos], m - ini + 1);
        seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[pos], fim - m);

        lazy[pos] = NEUTRAL_LAZY;
    }

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

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        lazy.push_back(-1);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        propagate(pos, ini, fim);
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

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

        if (ini >= p && fim <= q) {
            lazy[pos] = apply_lazy(lazy[pos], val, 1);
            seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);

            return;
        }

        int m = (ini + fim) >> 1;

        if (e[pos] == 0) e[pos] = create();
        update(e[pos], ini, m, p, q, val);

        if (d[pos] == 0) d[pos] = create();
        update(d[pos], m + 1, fim, p, q, val);

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

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

    void update(int p, int q, int val) {
        update(1, 1, n, p, q, val);
    }
};
```

9.13 Persistent

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

```cpp
// Update - update element at position id to a value val

// Problem:
// https://cses.fi/problemset/task/1737/

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

// How to use:
// vector<int> raiz(MAX); // vector to store the roots of each version
// Segtree seg = Segtree(INF);
// raiz[0] = seg.create(); // null node
// curr = 1; // keep track of the last version

// raiz[k] = seg.update(raiz[k], idx, val); // updating version k
// seg.query(raiz[k], l, r) // querying version k
// raiz[++curr] = raiz[k]; // create a new version based on version k

const int MAX = 2e5+17;
const int INF = 1e9+17;

typedef long long ftype;

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

    Segtree(int n) {
        this->n = n;
    }

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

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

    int update(int pos, int ini, int fim, int id, int val) {
        int novo = create();

        seg[novo] = seg[pos];
        e[novo] = e[pos];
        d[novo] = d[pos];

        if (ini == fim) {
            seg[novo] = val;
            return novo;
        }

        int m = (ini + fim) >> 1;

        if (id <= m) e[novo] = update(e[novo], ini, m, id, val);
        else d[novo] = update(d[novo], m + 1, fim, id, val);

        seg[novo] = f(seg[e[novo]], seg[d[novo]]);

        return novo;
    }

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

    int update(int pos, int id, int val) {
        return update(pos, 1, n, id, val);
    }
};
```