



Notebook - Maratona de Programação

Z-girls

Contents

1 DP	2	6 Grafos	7
1.1 Mochila	2	6.1 Kruskall	7
1.2 Troco Minimo	2	6.2 Dijkstra	8
1.3 Kadane	2	6.3 Dfs	8
1.4 Substr Palindromo	2	6.4 Bellman Ford	8
1.5 Moedas	2	6.5 Bipartite	8
1.6 Digitos	2	6.6 Floyd Warshall	8
2 Strings	3	6.7 Bfs	9
2.1 Kmp	3	6.8 Lca	9
2.2 Z-function	3	6.9 Dinic	9
3 Math	3	6.10 Find Cycle	10
3.1 Mdc	3	7 Template	10
3.2 Log	3	7.1 Template Clean	10
3.3 Divisores	3	7.2 Template	10
3.4 Sieve Of Eratosthenes	3	8 Algoritmos	10
3.5 Combinatoria	3	8.1 Ceil	10
3.6 Crt	3	8.2 Binary Search Last True	10
3.7 Prime Factors Sqrt	3	8.3 Kadane	11
3.8 Fatoracao Primos	4	8.4 Binary Exponentiation	11
3.9 Matrix Exponentiation	4	8.5 Delta-encoding	11
3.10 Fast Exponentiation	5	8.6 Binary Search First True	11
3.11 Mmc	5		
4 Misc	5		
4.1 Int128	5		
5 ED	5		
5.1 Seg Tree	5		
5.2 Seg Lazy	6		
5.3 Dsu	6		
5.4 Seg Pqru	7		

1 DP

1.1 Mochila

```
1 int val[MAXN], peso[MAXN], dp[MAXN][MAXS];
2
3 int knapsack(int n, int m){ // n Objetos | Peso max
4     for(int i=0; i<=n; i++){
5         for(int j=0; j<=m; j++){
6             if(i==0 or j==0)
7                 dp[i][j] = 0;
8             else if(peso[i-1]<=j)
9                 dp[i][j] = max(val[i-1]+dp[i-1][j-
10                     peso[i-1]], dp[i-1][j]);
11             else
12                 dp[i][j] = dp[i-1][j];
13         }
14     }
15     return dp[n][m];
16 }
```

1.2 Troco Minimo

```
1 int n;
2 vector<int> valores;
3
4 int tabela[1005];
5
6 int dp(int k){
7     if(k == 0){
8         return 0;
9     }
10    if(tabela[k] != -1)
11        return tabela[k];
12    int melhor = 1e9;
13    for(int i = 0; i < n; i++){
14        if(valores[i] <= k)
15            melhor = min(melhor, 1 + dp(k - valores[i
16                ]));
17    }
18    return tabela[k] = melhor;
19 }
```

1.3 Kadane

```
1 // achar uma subsequencia continua no array que a
2 // soma seja a maior possivel
3 // nesse caso vc precisa multiplicar exatamente 1
4 // elemento da subsequencia
5 // e achar a maior soma com isso
6
7 int n, x, arr[MAX], tab[MAX][2]; // tab[maior
8 // resposta no intervalo][foi multiplicado ou ão]
9
10 int dp(int i, bool mult) {
11     if (i == n-1) {
12         if (!mult) return arr[n-1]*x;
13         return arr[n-1];
14     }
15     if (tab[i][mult] != -1) return tab[i][mult];
16
17     int res;
18
19     if (mult) {
20         res = max(arr[i], arr[i] + dp(i+1, 1));
21     }
22     else {
23         res = max({
24             arr[i]*x,
25             arr[i]*x + dp(i+1, 1),
26             arr[i] + dp(i+1, 0)
27         });
28     }
29     tab[i][mult] = res;
30     return res;
31 }
```

```
24     });
25 }
26
27 return tab[i][mult] = res;
28 }
29
30 int main() {
31     memset(tab, -1, sizeof(tab));
32
33     int ans = -oo;
34     for (int i = 0; i < n; i++) {
35         ans = max(ans, dp(i, 0));
36     }
37
38     return 0;
39 }
40 }
```

1.4 Substr Palindromo

```
1 // êvoc deve informar se a substring de S formada
2 // pelos elementos entre os indices i e j
3 // é um palindromo ou ão.
4
5 char s[MAX];
6 int calculado[MAX][MAX]; // inciado com false, ou 0
7 int tabela[MAX][MAX];
8
9 int is_palin(int i, int j){
10     if(calculado[i][j]){
11         return tabela[i][j];
12     }
13     if(i == j) return true;
14     if(i + 1 == j) return s[i] == s[j];
15
16     int ans = false;
17     if(s[i] == s[j]){
18         if(is_palin(i+1, j-1)){
19             ans = true;
20         }
21     }
22     calculado[i][j] = true;
23     tabela[i][j] = ans;
24     return ans;
25 }
```

1.5 Moedas

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

1.6 Digtos

```
1 // achar a quantidade de numeros menores que R que
2 // possuem no maximo 3 digitos nao nulos
3 // a ideia eh utilizar da ordem lexicografica para
4 // checar isso pois se temos por exemplo
```

```

3 // o numero 8500, a gente sabe que se pegarmos o
  numero 7... qualquer digito depois do 7
4 // sera necessariamente menor q 8500
5
6 string r;
7 int tab[20][2][5];
8
9 // i - digito de R
10 // menor - ja pegou um numero menor que um digito de
   R
11 // qt - quantidade de digitos nao nulos
12 int dp(int i, bool menor, int qt){
13     if(qt > 3) return 0;
14     if(i >= r.size()) return 1;
15     if(tab[i][menor][qt] != -1) return tab[i][menor][
qt];
16
17     int dr = r[i] - '0';
18     int res = 0;
19
20     for(int d = 0; d <= 9; d++) {
21         int dnn = qt + (d > 0);
22         if(menor == true) {
23             res += dp(i+1, true, dnn);
24         }
25         else if(d < dr) {
26             res += dp(i+1, true, dnn);
27         }
28         else if(d == dr) {
29             res += dp(i+1, false, dnn);
30         }
31     }
32
33     return tab[i][menor][qt] = res;
34 }

```

2 Strings

2.1 Kmp

```

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

```

2.2 Z-function

```

1 vector<int> z_function(string s) {
2     int n = (int) s.length();
3     vector<int> z(n);
4     for (int i = 1, l = 0, r = 0; i < n; ++i) {
5         if (i <= r)
6             z[i] = min (r - i + 1, z[i - l]);
7         while (i + z[i] < n && s[z[i]] == s[i + z[i]
]])
8             ++z[i];
9         if (i + z[i] - 1 > r)
10            l = i, r = i + z[i] - 1;
11    }
12    return z;
13 }

```

3 Math

3.1 Mdc

```

1 long long gcd(long long a, long long b){
2     return b ? gcd(b, a % b) : a;
3 }
4
5 // or just use __gcd(a,b)

```

3.2 Log

```

1 int intlog(double base, double x) {
2     return (int)(log(x) / log(base));
3 }

```

3.3 Divisores

```

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

```

3.4 Sieve Of Eratosthenes

```

1 int n;
2 vector<bool> is_prime(n+1, true);
3 is_prime[0] = is_prime[1] = false;
4 for (int i = 2; i <= n; i++) {
5     if (is_prime[i] && (long long)i * i <= n) {
6         for (int j = i * i; j <= n; j += i)
7             is_prime[j] = false;
8     }
9 }

```

3.5 Combinatoria

```

1 int comb(int k){
2     if(k==1 or k==0) return 0;
3     return (k*(k-1))/2;
4 }

```

3.6 Crt

```

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

```

3.7 Prime Factors Sqrt

```

1 map<ll, ll> expo;
2
3 void primeFactors(ll n) {
4     while (n % 2 == 0) {
5         expo[2]++;
6         n = n/2;
7     }
8
9     for (ll i = 3; i <= sqrt(n); i = i + 2) {
10        while (n % i == 0) {
11            expo[i]++;
12            n = n/i;
13        }
14    }
15
16    if (n > 2) expo[n]++;
17 }

```

3.8 Fatoracao Primos

```

1 vector<pair<int, int>> fatora(int x) {
2     map<int, int> expoentes;
3     while(x > 1) {
4         expoentes[lp[x]]++; // aumentamos o expoente do
5         // primo lp[x] em 1 na resposta
6         x /= lp[x];
7     }
8     vector<pair<int, int>> ans;
9     for(pair<int, int> p : expoentes)
10        ans.emplace_back(p);
11    return ans;
12 }

```

3.9 Matrix Exponentiation

```

1 #include <bits/stdc++.h>
2 #define debug(x) cout << "[" << #x << " = " << x << "
3 ] "
4 #define ff first
5 #define ss second
6 using namespace std;
7 using ll = long long;
8 using ld = long double;
9 using pii = pair<int,int>;
10 using vi = vector<int>;
11
12 using tii = tuple<int,int,int>;
13 // auto [a,b,c] = ...
14 // .insert({a,b,c})
15
16 const int oo = (int)1e9; //INF to INT
17 const ll OO = 0x3f3f3f3f3f3f3fLL; //INF to LL
18
19 /*wa? coloca long long que passa;
20 testar casos, n = 0? n = 1? todos os numeros iguais?
21 Uma resposta ótima pode ter tamanho 2?
22 RELER O ENUNCIADO!*/
23
24 const int MOD = 1e9+7;
25
26 struct Mat{
27     vector<vector<ll>> matriz;
28     int l, c;
29
30     Mat(vector<vector<ll>>& mat){
31         matriz = mat;
32         l = mat.size();
33         c = mat[0].size();
34     }
35
36     Mat(int r, int col, bool identidade=false){

```

```

37         //qnt linhas, qnt colunas, identidade
38         l = r; c = col;
39         matriz.assign(l, vector<ll>(col, 0));
40         if(identidade){
41             for(int i = 0; i < min(l,col); i++){
42                 matriz[i][i] = 1;
43             }
44         }
45
46     Mat operator * (const Mat& a) const{
47         assert(c == a.l); //qnt lcolunas mat deve ser
48         //igual qnt linhas a
49         vector<vector<ll>> resp(l, vector<ll>(a.c, 0)
50 );
51         //multiplica. Algoritmo úcbico.
52         for(int i = 0; i < l; i++){
53             for(int j = 0; j < a.c; j++){
54                 for(int k = 0; k < a.l; k++){
55                     resp[i][j] = (resp[i][j] + (
56                         matriz[i][k]*a.matriz[k][j]) % MOD) % MOD;
57                 }
58             }
59         }
60         return Mat(resp);
61     }
62
63     Mat operator + (const Mat& a) const{
64         assert(l == a.l && c == a.c); //dimensoes
65         //iguais
66         vector<vector<ll>> resp(l, vector<ll>(c,0));
67         for(int i = 0; i < l; i++){
68             for(int j = 0; j < c; j++){
69                 resp[i][j] = (resp[i][j] + matriz[i][
70                     j] + a.matriz[i][j]) % MOD;
71             }
72         }
73         return Mat(resp);
74     }
75 };
76
77 Mat fexp(Mat& base, ll expoente, ll sz){
78     Mat result = Mat(sz, sz, 1);
79     while(expoente > 0){
80         if(expoente & 1) result = result * base;
81         base = base * base;
82         expoente /= 2;
83     }
84     return result;
85 }
86
87 int main() {
88     ios::sync_with_stdio(false);
89     cin.tie(NULL);
90
91     ll n, a, b;
92     cin >> a >> b >> n;
93
94     Mat X(2,2);
95
96     //f_i = c1 * f_(i-1) + c2 * f_(i-2) + ... + ck * f
97     //(i-k)
98     // monta a matriz X
99     // A *2 diagonal (todas as cōposies acima dos
100     // elementos q pertecem a diagonal principal) = 1
101     // A ultima linha é composta por c_k, c_(k-1),
102     // c_(k-2), ..., c_2, c_1
103     //Para se ter o pē-simo elemento é ós fazer X^(P
104     // -1) pq indexa em 0
105     //e multiplicar pela matriz coluna, onde os
106     //elementos ãso: [f(0)
107     //
108     //f(1)

```

```

99         f(2)
100         ....
101         f(k-1)
102     ]
103
104     */
105
106     //nessa ãquesto a gente tem que f_i = f_(i-1) - f
107     (i-2), sendo que f_0 = a e f_1 = b, a matriz fica
108
109     // 0 1
110     // -1 1
111     X.matriz[0][1] = 1;
112     X.matriz[1][0] = -1;
113     X.matriz[1][1] = 1;
114
115     Mat y = fexp(X,n-1,2);
116
117     ll ans = y.matriz[0][0] * a + y.matriz[0][1] * b;
118
119     while(ans < 0)
120         ans += MOD;
121
122     cout << ans % MOD << endl;
123 }

```

3.10 Fast Exponentiation

```

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

```

3.11 Mmc

```

1 long long lcm(long long a, long long b){
2     return (a/__gcd(a,b)*b);
3 }

```

4 Misc

4.1 Int128

```

1 __int128 read() {
2     __int128 x = 0, f = 1;
3     char ch = getchar();
4     while (ch < '0' || ch > '9') {
5         if (ch == '-') f = -1;
6         ch = getchar();
7     }
8     while (ch >= '0' && ch <= '9') {
9         x = x * 10 + ch - '0';
10        ch = getchar();
11    }
12    return x * f;
13 }
14 void print(__int128 x) {
15     if (x < 0) {
16         putchar('-');
17         x = -x;
18     }
19     if (x > 9) print(x / 10);

```

```

20     putchar(x % 10 + '0');
21 }

```

5 ED

5.1 Seg Tree

```

1 class SegTree{
2     vector<int> seg;
3     vector<int> v;
4     int size;
5     int el_neutro = INT_MAX;
6
7     int f(int a, int b){
8         return min(a,b);
9     }
10
11 void update(int pos, int ini, int fim, int i, int
12 val){
13     if(i < ini or i > fim) return;
14     if(ini == fim){
15         seg[pos] = val; return;
16     }
17
18     int m = (ini+fim)/2;
19     int e = 2*pos, d = 2*pos+1;
20     update(e, ini, m, i, val);
21     update(d, m+1, fim, i, val);
22
23     seg[pos] = f(seg[e], seg[d]);
24 }
25
26 int query(int pos, int ini, int fim, int p, int q
27 ){
28     if(q < ini or p > fim) return el_neutro;
29     if(p <= ini and fim <= q) return seg[pos];
30
31     int m = (ini + fim)/2;
32     int e = 2*pos, d = 2*pos+1;
33     return f(query(e,ini,m,p,q), query(d,m+1,fim,
34 p,q));
35 }
36
37 void build(int pos, int ini, int fim){
38     if(ini == fim){
39         seg[pos] = v[ini]; return;
40     }
41
42     int m = (ini+fim)/2;
43     int e = 2*pos, d=2*pos+1;
44
45     build(e,ini,m);
46     build(d,m+1,fim);
47
48     seg[pos] = f(seg[e], seg[d]);
49 }
50
51 public:
52     SegTree(int n, vector<int> source): seg(4*
53 size), v(size){
54         size = n;
55         for(int i=0; i<size; i++) v[i] = source[i]
56 ];
57     }
58
59     void update(int i, int val){ return update
60 (1,1,size,i,val); }
61
62     int query(int p, int q){ return query(1,1,
63 size,p,q); }
64
65     void build(){ return build(1,1,size); }

```

```
59 };
```

5.2 Seg Lazy

```
1 using ll = long long;
2
3 struct segTree {
4     int size;
5     vector<ll> tree, lazy;
6
7     ll modify_op(ll a, ll b, ll len) {
8         if (b == -1) return a;
9         return b * len;
10    }
11
12    void apply_mod_op(ll &a, ll b, ll len) {
13        a = modify_op(a, b, len);
14    }
15
16    ll merge(ll a, ll b) {
17        return a + b;
18    }
19
20    void init(int n) {
21        size = 1;
22        while (size < n) size *= 2;
23        tree.assign(2 * size, 0LL);
24        lazy.assign(2 * size, -1);
25    }
26
27    void propagate(int x, int lx, int rx) {
28        if (rx - lx == 1) return;
29
30        int m = (lx + rx) / 2;
31        apply_mod_op(lazy[2 * x + 1], lazy[x], 1);
32        apply_mod_op(tree[2 * x + 1], lazy[x], m - lx);
33    };
34
35    apply_mod_op(lazy[2 * x + 2], lazy[x], 1);
36    apply_mod_op(tree[2 * x + 2], lazy[x], rx - m);
37
38    lazy[x] = -1;
39
40    void build(vector<int> &arr, int x, int lx, int rx) {
41        if (rx - lx == 1) {
42            if (lx < (int)arr.size())
43                tree[x] = arr[lx];
44
45            return;
46        }
47
48        int m = (lx + rx) / 2;
49        build(arr, 2 * x + 1, lx, m);
50        build(arr, 2 * x + 2, m, rx);
51
52        tree[x] = merge(tree[2 * x + 1], tree[2 * x + 2]);
53    }
54
55    void build(vector<int> &arr) {
56        build(arr, 0, 0, size);
57    }
58
59    void update(int l, int r, int v, int x, int lx, int rx) {
60        propagate(x, lx, rx);
61
62        if (lx >= r || l >= rx) return;
63        if (lx >= l && rx <= r) {
64            apply_mod_op(lazy[x], v, 1);
```

```
65            apply_mod_op(tree[x], v, rx - lx);
66            return;
67        }
68
69        int m = (lx + rx) / 2;
70        update(l, r, v, 2 * x + 1, lx, m);
71        update(l, r, v, 2 * x + 2, m, rx);
72
73        tree[x] = merge(tree[2 * x + 1], tree[2 * x + 2]);
74    }
75
76    void update(int l, int r, int v) {
77        update(l, r, v, 0, 0, size);
78    }
79
80    ll query(int l, int r, int x, int lx, int rx) {
81        propagate(x, lx, rx);
82
83        if (lx >= r || l >= rx) return 0;
84        if (lx >= l && rx <= r) return tree[x];
85
86        int m = (lx + rx) / 2;
87        ll s1 = query(l, r, 2 * x + 1, lx, m);
88        ll s2 = query(l, r, 2 * x + 2, m, rx);
89
90        return merge(s1, s2);
91    }
92
93    ll query(int l, int r) {
94        return query(l, r, 0, 0, size);
95    }
96
97    void debug() {
98        for (auto e : tree)
99            cout << e << ' ';
100        cout << endl;
101
102        for (auto e : lazy)
103            cout << e << ' ';
104        cout << endl;
105    }
106 };
```

5.3 Dsu

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX = 1e6+17;
6
7 struct DSU {
8     int n;
9     vector<int> link, sizes;
10
11     DSU(int n) {
12         this->n = n;
13         link.assign(n+1, 0);
14         sizes.assign(n+1, 1);
15
16         for (int i = 0; i <= n; i++)
17             link[i] = i;
18     }
19
20     int find(int x) {
21         while (x != link[x])
22             x = link[x];
23
24         return x;
25     }
26
27     bool same(int a, int b) {
```

```

28     return find(a) == find(b);
29 }
30
31 void unite(int a, int b) {
32     a = find(a);
33     b = find(b);
34
35     if (a == b) return;
36
37     if (sizes[a] < sizes[b])
38         swap(a, b);
39
40     sizes[a] += sizes[b];
41     link[b] = a;
42 }
43
44 int size(int x) {
45     return sizes[x];
46 }
47 };
48
49 int main() {
50     ios::sync_with_stdio(false);
51     cin.tie(NULL);
52
53     int cities, roads; cin >> cities >> roads;
54     vector<int> final_roads;
55     int ans = 0;
56     DSU dsu = DSU(cities);
57     for (int i = 0, a, b; i < roads; i++) {
58         cin >> a >> b;
59         dsu.unite(a, b);
60     }
61
62     for (int i = 2; i <= cities; i++) {
63         if (!dsu.same(1, i)) {
64             ans++;
65             final_roads.push_back(i);
66             dsu.unite(1, i);
67         }
68     }
69
70     cout << ans << '\n';
71     for (auto e : final_roads) {
72         cout << "1 " << e << '\n';
73     }
74
75 }

```

5.4 Seg Pqru

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 class SegTree{
5     vector<int> seg;
6     vector<int> v;
7     int size;
8     int el_neutro = INT_MAX;
9
10    int f(int a, int b){
11        return min(a,b);
12    }
13
14    void update_range(int pos, int ini, int fim, int
15    l, int r, int val){
16        if(r < ini or l > fim) return;
17        if(l <= ini and fim <= r){
18            seg[pos] += val;
19        }
20
21        int mid = (ini+fim)/2;

```

```

22        update_range(2*pos, ini, mid, l, r, val);
23        update_range(2*pos+1, mid+1, fim, l, r, val);
24    }
25
26    int query_point(int pos, int ini, int fim, int i)
27    {
28        if(ini == fim) return seg[pos];
29
30        int mid = (ini + fim)/2;
31        if(i<=mid)
32            return query_point(2*pos, ini, mid, i);
33        else
34            return query_point(2*pos+1, mid+1, fim, i
35    );
36    }
37
38    void build(int pos, int ini, int fim){
39        if(ini == fim){
40            seg[pos] = v[ini]; return;
41        }
42
43        int m = (ini+fim)/2;
44        int e = 2*pos, d=2*pos+1;
45
46        build(e,ini,m);
47        build(d,m+1,fim);
48
49        seg[pos] = f(seg[e], seg[d]);
50    }
51
52    public:
53    SegTree(int n, vector<int> source): seg(4*size),
54    v(size){
55        size = n;
56        for(int i=0; i<size; i++) v[i] = source[i];
57    }
58
59    void update(int l, int r, int val){ return
60    update_range(1,1,size,l, r,val); }
61
62    int query(int i){ return query_point(1,1,size,i);
63    }
64
65    void build(){ return build(1,1,size); }
66 };

```

6 Grafos

6.1 Kruskall

```

1 vector<int> parent, rank;
2
3 void make_set(int v) {
4     parent[v] = v;
5     rank[v] = 0;
6 }
7
8 int find_set(int v) {
9     if (v == parent[v])
10         return v;
11     return parent[v] = find_set(parent[v]);
12 }
13
14 void union_sets(int a, int b) {
15     a = find_set(a);
16     b = find_set(b);
17     if (a != b) {
18         if (rank[a] < rank[b])
19             swap(a, b);
20         parent[b] = a;
21         if (rank[a] == rank[b])
22             rank[a]++;

```

```

23     }
24 }
25
26 struct Edge {
27     int u, v, weight;
28     bool operator<(Edge const& other) {
29         return weight < other.weight;
30     }
31 };
32
33 int n;
34 vector<Edge> edges;
35
36 int cost = 0;
37 vector<Edge> result;
38 parent.resize(n);
39 rank.resize(n);
40 for (int i = 0; i < n; i++)
41     make_set(i);
42
43 sort(edges.begin(), edges.end());
44
45 for (Edge e : edges) {
46     if (find_set(e.u) != find_set(e.v)) {
47         cost += e.weight;
48         result.push_back(e);
49         union_sets(e.u, e.v);
50     }
51 }

```

6.2 Dijkstra

```

1  const int INF = 1000000000;
2  vector<vector<pair<int, int>>> adj;
3
4  void dijkstra(int s, vector<int> & d, vector<int> & p
5  ) {
6      int n = adj.size();
7      d.assign(n, INF);
8      p.assign(n, -1);
9
10     d[s] = 0;
11     set<pair<int, int>> q;
12     q.insert({0, s});
13     while (!q.empty()) {
14         int v = q.begin()->second;
15         q.erase(q.begin());
16
17         for (auto edge : adj[v]) {
18             int to = edge.first;
19             int len = edge.second;
20
21             if (d[v] + len < d[to]) {
22                 q.erase({d[to], to});
23                 d[to] = d[v] + len;
24                 p[to] = v;
25                 q.insert({d[to], to});
26             }
27         }
28     }

```

6.3 Dfs

```

1  vector<vector<int>> graph;
2  vector<bool> visited;
3
4  void dfs(int vertex){
5      visited[vertex] = true;
6
7      for(int w: graph[vertex]){
8          if(!visited[w]){

```

```

9          dfs(w);
10     }
11 }
12 }

```

6.4 Bellman Ford

```

1  struct edge
2  {
3      int a, b, cost;
4  };
5
6  int n, m, v;
7  vector<edge> e;
8  const int INF = 1000000000;
9
10 void solve()
11 {
12     vector<int> d (n, INF);
13     d[v] = 0;
14     for (int i=0; i<n-1; ++i)
15         for (int j=0; j<m; ++j)
16             if (d[e[j].a] < INF)
17                 d[e[j].b] = min (d[e[j].b], d[e[j].a]
18                     + e[j].cost);
19 }

```

6.5 Bipartite

```

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

```

6.6 Floyd Warshall

```

1  for (int k = 0; k < n; ++k) {
2      for (int i = 0; i < n; ++i) {
3          for (int j = 0; j < n; ++j) {
4              if (d[i][k] < INF && d[k][j] < INF)

```



```

5         d[i][j] = min(d[i][j], d[i][k] + d[k
6         ][j]);
7     }
8 }

```

6.7 Bfs

```

1 void bfs(int start){
2
3     queue<int> q;
4     q.push(start);
5
6     vector<bool> visited(GRAPH_MAX_SIZE, false);
7     visited[start] = true;
8     while(q.size()){
9         int u = q.front();
10        q.pop();
11        for(int w: graph[u]){
12            if(not visited[w]){
13                q.push(w);
14                visited[w] = true;
15            }
16        }
17    }
18 }
19 }

```

6.8 Lca

```

1 const int MAX = 2e5+17;
2
3 int n, l;
4 vector<vector<int>> adj;
5 // vector<pair<int, int>> adj[MAX];
6 // int dist[MAX];
7
8 int timer;
9 vector<int> tin, tout;
10 vector<vector<int>> up;
11
12 void dfs(int v, int p)
13 {
14     tin[v] = ++timer;
15     up[v][0] = p;
16     for (int i = 1; i <= l; ++i)
17         up[v][i] = up[up[v][i-1]][i-1];
18
19     for (int u : adj[v]) {
20         if (u != p)
21             dfs(u, v);
22     }
23
24     /*for (auto [u, peso] : adj[v]) {
25         if (u != p) {
26             dist[u] = dist[v] + peso;
27             dfs(u, v);
28         }
29     }*/
30
31     tout[v] = ++timer;
32 }
33
34 bool is_ancestor(int u, int v)
35 {
36     return tin[u] <= tin[v] && tout[u] >= tout[v];
37 }
38
39 int lca(int u, int v)
40 {
41     if (is_ancestor(u, v))
42         return u;

```

```

43     if (is_ancestor(v, u))
44         return v;
45     for (int i = l; i >= 0; --i) {
46         if (!is_ancestor(up[u][i], v))
47             u = up[u][i];
48     }
49     return up[u][0];
50 }
51
52 void preprocess(int root) {
53     tin.resize(MAX);
54     tout.resize(MAX);
55     timer = 0;
56     up.assign(MAX, vector<int>(32));
57     dfs(root, root);
58 }
59
60 //distance between a and b
61 // dist[a] + dist[b] - 2*dist[lca(a, b)]

```

6.9 Dinic

```

1 const int N = 300;
2
3 struct Dinic {
4     struct Edge{
5         int from, to; ll flow, cap;
6     };
7     vector<Edge> edge;
8
9     vector<int> g[N];
10    int ne = 0;
11    int lvl[N], vis[N], pass;
12    int qu[N], px[N], qt;
13
14    ll run(int s, int sink, ll minE) {
15        if(s == sink) return minE;
16
17        ll ans = 0;
18
19        for(; px[s] < (int)g[s].size(); px[s]++) {
20            int e = g[s][px[s]];
21            auto &v = edge[e], &rev = edge[e^1];
22            if(lvl[v.to] != lvl[s]+1 || v.flow >= v.
23                cap) continue; // v.cap - v.flow
24            < lim
25            ll tmp = run(v.to, sink, min(minE, v.cap - v
26                .flow));
27            v.flow += tmp, rev.flow -= tmp;
28            ans += tmp, minE -= tmp;
29            if(minE == 0) break;
30        }
31        return ans;
32    }
33
34    bool bfs(int source, int sink) {
35        qt = 0;
36        qu[qt++] = source;
37        lvl[source] = 1;
38        vis[source] = ++pass;
39        for(int i = 0; i < qt; i++) {
40            int u = qu[i];
41            px[u] = 0;
42            if(u == sink) return true;
43            for(auto& ed : g[u]) {
44                auto v = edge[ed];
45                if(v.flow >= v.cap || vis[v.to] ==
46                    pass) continue; // v.cap - v.flow < lim
47                vis[v.to] = pass;
48                lvl[v.to] = lvl[u]+1;
49                qu[qt++] = v.to;
50            }

```

```

48     }
49     return false;
50 }
51 ll flow(int source, int sink) {
52     reset_flow();
53     ll ans = 0;
54     //for(lim = (1LL << 62); lim >= 1; lim /= 2)
55     while(bfs(source, sink))
56         ans += run(source, sink, LLINF);
57     return ans;
58 }
59 void addEdge(int u, int v, ll c, ll rc) {
60     Edge e = {u, v, 0, c};
61     edge.pb(e);
62     g[u].push_back(ne++);
63
64     e = {v, u, 0, rc};
65     edge.pb(e);
66     g[v].push_back(ne++);
67 }
68 void reset_flow() {
69     for(int i = 0; i < ne; i++)
70         edge[i].flow = 0;
71     memset(lvl, 0, sizeof(lvl));
72     memset(vis, 0, sizeof(vis));
73     memset(qu, 0, sizeof(qu));
74     memset(px, 0, sizeof(px));
75     qt = 0; pass = 0;
76 }
77 };

```

6.10 Find Cycle

```

1  bitset<MAX> visited;
2  vector<int> path;
3  vector<int> adj[MAX];
4
5  bool dfs(int u, int p){
6
7      if (visited[u]) return false;
8
9      path.pb(u);
10     visited[u] = true;
11
12     for (auto v : adj[u]){
13         if (visited[v] and u != v and p != v){
14             path.pb(v); return true;
15         }
16
17         if (dfs(v, u)) return true;
18     }
19
20     path.pop_back();
21     return false;
22 }
23
24 bool has_cycle(int N){
25
26     visited.reset();
27
28     for (int u = 1; u <= N; ++u){
29         path.clear();
30         if (not visited[u] and dfs(u,-1))
31             return true;
32
33     }
34
35     return false;
36 }

```

7 Template

7.1 Template Clean

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  // g++ -std=c++20 main.cpp
5
6  // g++ -std=c++17 -Wshadow -Wall -Wextra -Wformat=2 -
   Wconversion -fsanitize=address,undefined -fno-
   sanitize-recover -Wfatal-errors
7
8  // cout << fixed << setprecision(12) << value << endl
   ;
9
10 // freopen("input.txt", "r", stdin);
11 // freopen("output.txt", "w", stdout);
12
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(NULL);
16
17
18     return 0;
19 }
20 }

```

7.2 Template

```

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

```

8 Algoritmos

8.1 Ceil

```

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

```

8.2 Binary Search Last True

```

1  int last_true(int lo, int hi, function<bool(int)> f)
   {
2      lo--;

```

```

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

```

8.3 Kadane

```

1 int ans = a[0], ans_l = 0, ans_r = 0;
2 int sum = 0, minus_pos = -1;
3
4 for (int r = 0; r < n; ++r) {
5     sum += a[r];
6     if (sum > ans) {
7         ans = sum;
8         ans_l = minus_pos + 1;
9         ans_r = r;
10    }
11    if (sum < 0) {
12        sum = 0;
13        minus_pos = r;
14    }
15 }

```

8.4 Binary Exponentiation

```

1 long long power(long long a, long long b) {
2     long long res = 1;
3     while (b > 0) {
4         if (b & 1)
5             res = res * a;
6         a = a * a;
7         b >>= 1;
8     }
9     return res;
10 }

```

8.5 Delta-encoding

```

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

```

```

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

```

8.6 Binary Search First True

```

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

```