



# Notebook - Maratona de Programação

Py tá O(N)

## Contents

<b>1</b>	<b>Algoritmos</b>	<b>2</b>		
1.1	Binary Exponentiation . . . . .	2		
1.2	Binary Search First True . . . . .	2		
1.3	Binary Search Last True . . . . .	2		
1.4	Delta-encoding . . . . .	2		
1.5	Kadane . . . . .	2		
<b>2</b>	<b>DP</b>	<b>2</b>		
2.1	Digitos . . . . .	2		
2.2	Kadane . . . . .	2		
2.3	Mochila . . . . .	3		
2.4	Moedas . . . . .	3		
2.5	Substr Palindromo . . . . .	3		
2.6	Troco Minimo . . . . .	3		
<b>3</b>	<b>ED</b>	<b>3</b>		
3.1	Dsu . . . . .	3		
3.2	Seg Pqru . . . . .	4		
3.3	Seg Tree . . . . .	4		
<b>4</b>	<b>Grafos</b>	<b>5</b>		
4.1	Bellman Ford . . . . .	5		
4.2	Bfs . . . . .	5		
4.3	Bipartite . . . . .	5		
4.4	Dfs . . . . .	6		
4.5	Dijkstra . . . . .	6		
4.6	Find Cycle . . . . .	6		
4.7	Floyd Warshall . . . . .	6		
4.8	Kruskall . . . . .	6		
4.9	Lca . . . . .	7		
<b>5</b>	<b>Math</b>	<b>7</b>		
5.1	Combinatoria . . . . .	7		
5.2	Divisores . . . . .	7		
5.3	Fast Exponentiation . . . . .	7		
5.4	Fatoracao Primos . . . . .	7		
5.5	Mdc . . . . .	7		
			5.6	Mmc . . . . . 7
			5.7	Sieve Of Eratosthenes . . . . . 7
<b>6</b>	<b>Strings</b>	<b>8</b>		
6.1	Kmp . . . . .	8		
6.2	Z-function . . . . .	8		
<b>7</b>	<b>Template</b>	<b>8</b>		
7.1	Template . . . . .	8		

# 1 Algoritmos

## 1.1 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 }
```

## 1.2 Binary Search First True

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

## 1.3 Binary Search Last True

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

## 1.4 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 }
```

## 1.5 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 }
```

## 2 DP

### 2.1 Dígitos

```
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
5 // o numero 8500, a gente sabe que se pegarmos o
6 // numero 7... qualquer digito depois do 7
7 // sera necessariamente menor q 8500
8
9 string r;
10 int tab[20][2][5];
11
12 // i - digito de R
13 // menor - ja pegou um numero menor que um digito de
14 // R
15 // qt - quantidade de digitos nao nulos
16 int dp(int i, bool menor, int qt){
17     if(qt > 3) return 0;
18     if(i >= r.size()) return 1;
19     if(tab[i][menor][qt] != -1) return tab[i][menor][qt];
20
21     int dr = r[i] - '0';
22     int res = 0;
23
24     for(int d = 0; d <= 9; d++) {
25         int dnn = qt + (d > 0);
26         if(menor == true) {
27             res += dp(i+1, true, dnn);
28         }
29         else if(d < dr) {
30             res += dp(i+1, true, dnn);
31         }
32         else if(d == dr) {
33             res += dp(i+1, false, dnn);
34         }
35     }
36
37     return tab[i][menor][qt] = res;
38 }
```

### 2.2 Kadane

```

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

```

## 2.3 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 }

```

## 2.4 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 }

```

## 2.5 Substr Palindromo

```

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

```

## 2.6 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 }

```

## 3 ED

### 3.1 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 }

```

### 3.2 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
23        update_range(2*pos, ini, mid, l, r, val);
24        update_range(2*pos+1, mid+1, fim, l, r, val);
25    }
26
27    int query_point(int pos, int ini, int fim, int i)
28    {
29        if(ini == fim) return seg[pos];
30
31        int mid = (ini + fim)/2;
32        if(i<=mid)
33            return query_point(2*pos, ini, mid, i);
34        else
35            return query_point(2*pos+1, mid+1, fim, i
36        );
37    }
38
39    void build(int pos, int ini, int fim){
40        if(ini == fim){
41            seg[pos] = v[ini]; return;
42        }
43
44        int m = (ini+fim)/2;
45        int e = 2*pos, d=2*pos+1;
46
47        build(e,ini,m);
48        build(d,m+1,fim);
49
50        seg[pos] = f(seg[e], seg[d]);
51    }
52
53    public:
54    SegTree(int n, vector<int> source): seg(4*size),
55    v(size){
56        size = n;
57        for(int i=0; i<size; i++) v[i] = source[i];
58    }
59
60    void update(int l, int r, int val){ return
61    update_range(1,1,size,l, r,val); }
62
63    int query(int i){ return query_point(1,1,size,i);
64    }
65
66    void build(){ return build(1,1,size); }
67 };

```

### 3.3 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
    val){
12     if(i < ini or i > fim) return;
13     if(ini == fim){
14         seg[pos] = val; return;
15     }
16
17     int m = (ini+fim)/2;
18     int e = 2*pos, d = 2*pos+1;
19     update(e, ini, m, i, val);
20     update(d, m+1, fim, i, val);
21
22     seg[pos] = f(seg[e], seg[d]);
23 }
24
25 int query(int pos, int ini, int fim, int p, int q
    ){
26     if(q < ini or p > fim) return el_neutro;
27     if(p <= ini and fim <= q) return seg[pos];
28
29     int m = (ini + fim)/2;
30     int e = 2*pos, d = 2*pos+1;
31     return f(query(e,ini,m,p,q), query(d,m+1,fim,
    p,q));
32 }
33
34 void build(int pos, int ini, int fim){
35     if(ini == fim){
36         seg[pos] = v[ini]; return;
37     }
38
39     int m = (ini+fim)/2;
40     int e = 2*pos, d=2*pos+1;
41
42     build(e,ini,m);
43     build(d,m+1,fim);
44
45     seg[pos] = f(seg[e], seg[d]);
46 }
47
48 public:
49     SegTree(int n, vector<int> source): seg(4*
    size), v(size){
50         size = n;
51         for(int i=0; i<size; i++) v[i] = source[i
    ];
52     }
53
54     void update(int i, int val){ return update
    (1,1,size,i,val); }
55
56     int query(int p, int q){ return query(1,1,
    size,p,q); }
57
58     void build(){ return build(1,1,size); }
59 };

```

## 4 Grafos

### 4.1 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]
    + e[j].cost);
18 }

```

### 4.2 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 }

```

### 4.3 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 }

```

## 4.4 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 }

```

## 4.5 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    }

```

## 4.6 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    return false;
35 }
36 }

```

## 4.7 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     }

```

## 4.8 Kruskal

```

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     }

```

## 4.9 Lca

```

1  int n, l;
2  vector<vector<int>> adj;
3
4  int timer;
5  vector<int> tin, tout;
6  vector<vector<int>> up;
7
8  void dfs(int v, int p)
9  {
10     tin[v] = ++timer;
11     up[v][0] = p;
12     for (int i = 1; i <= l; ++i)
13         up[v][i] = up[up[v][i-1]][i-1];
14
15     for (int u : adj[v]) {
16         if (u != p)
17             dfs(u, v);
18     }
19
20     tout[v] = ++timer;
21 }
22
23 bool is_ancestor(int u, int v)
24 {
25     return tin[u] <= tin[v] && tout[u] >= tout[v];
26 }
27
28 int lca(int u, int v)
29 {
30     if (is_ancestor(u, v))
31         return u;
32     if (is_ancestor(v, u))
33         return v;
34     for (int i = l; i >= 0; --i) {
35         if (!is_ancestor(up[u][i], v))
36             u = up[u][i];
37     }
38     return up[u][0];
39 }
40
41 void preprocess(int root) {
42     tin.resize(n);
43     tout.resize(n);
44     timer = 0;
45     l = ceil(log2(n));
46     up.assign(n, vector<int>(l + 1));
47     dfs(root, root);
48 }

```

# 5 Math

## 5.1 Combinatoria

```

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

```

## 5.2 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 }

```

## 5.3 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 }

```

## 5.4 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;

```

## 5.5 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)

```

## 5.6 Mmc

```

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

```

## 5.7 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  }

```

## 6 Strings

### 6.1 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 }
```

### 6.2 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 }
```

13 }

## 7 Template

### 7.1 Template

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