

# Competitive Programming Notebook

Programadores Roblox

## Contents

<b>1</b>	<b>DP</b>	<b>2</b>
1.1	Lis . . . . .	2
1.2	Lcs . . . . .	2
1.3	Knapsack . . . . .	2
<b>2</b>	<b>String</b>	<b>2</b>
<b>3</b>	<b>Geometry</b>	<b>2</b>
<b>4</b>	<b>Graph</b>	<b>2</b>
4.1	Dijkstra . . . . .	2
<b>5</b>	<b>Math</b>	<b>2</b>
5.1	Exgcd . . . . .	2
5.2	Fexp . . . . .	2
5.3	Equacao Diofantina . . . . .	2
<b>6</b>	<b>DS</b>	<b>2</b>
6.1	Oset . . . . .	2
6.2	Dsu . . . . .	3
<b>7</b>	<b>Primitives</b>	<b>3</b>
<b>8</b>	<b>General</b>	<b>3</b>
8.1	Bitwise . . . . .	3

# 1 DP

## 1.1 Lis

## 1.2 Lcs

## 1.3 Knapsack

```

1 // dp[i][j] => i-esimo item com j-carga sobrando na
  mochila
2 // O(N * W)
3
4 for(int j = 0; j < MAXN; j++) {
5     dp[0][j] = 0;
6 }
7 for(int i = 1; i <= N; i++) {
8     for(int j = 0; j <= W; j++) {
9         if(items[i].first > j) {
10             dp[i][j] = dp[i-1][j];
11         }
12         else {
13             dp[i][j] = max(dp[i-1][j], dp[i-1][j-
14 items[i].first] + items[i].second);
15         }
16 }

```

# 2 String

# 3 Geometry

# 4 Graph

## 4.1 Dijkstra

```

1 // SSP com pesos positivos.
2 // O((V + E) log V).
3
4 vector<int> dijkstra(int S) {
5     vector<bool> vis(MAXN, 0);
6     vector<ll> dist(MAXN, LLONG_MAX);
7     dist[S] = 0;
8     priority_queue<pii, vector<pii>, greater<pii>> pq
9     ;
10    pq.push({0, S});
11    while(pq.size()) {
12        ll v = pq.top().second;
13        pq.pop();
14        if(vis[v]) continue;
15        vis[v] = 1;
16        for(auto &[peso, vizinho] : adj[v]) {
17            if(dist[vizinho] > dist[v] + peso) {
18                dist[vizinho] = dist[v] + peso;
19                pq.push({dist[vizinho], vizinho});
20            }
21        }
22    }
23    return dist;
24 }

```

# 5 Math

## 5.1 Exgcd

```

1 // 0 retorno da funcao eh {n, m, g}
2 // e significa que gcd(a, b) = g e
3 // n e m sao inteiros tais que an + bm = g
4 array<ll, 3> exgcd(int a, int b) {
5     if(b == 0) return {1, 0, a};
6     auto [m, n, g] = exgcd(b, a % b);
7     return {n, m - a / b * n, g};
8 }

```

## 5.2 Fexp

```

1 // a^e mod m
2 // O(log n)
3
4 ll fexp(ll a, ll e, ll m) {
5     a %= m;
6     ll ans = 1;
7     while (e > 0) {
8         if (e & 1) ans = ansa % m;
9         a = aa % m;
10        e /= 2;
11    }
12    return ans%m;
13 }

```

## 5.3 Equacao Diofantina

```

1 // resolve equacao ax + by = c
2 // retorno {existe sol., x, y, g}
3 array<ll, 4> find_any_solution(ll a, ll b, ll c) {
4     auto [x, y, g] = exgcd(a, b);
5     if (c % g) return {false, 0, 0, 0};
6     x *= c / g;
7     y *= c / g;
8     return {true, x, y, g};
9 }

```

# 6 DS

## 6.1 Oset

```

1
2 #include<ext/pb_ds/assoc_container.hpp>
3 #include<ext/pb_ds/tree_policy.hpp>
4 using namespace __gnu_pbds;
5 using namespace std;
6
7 template<typename T> using ordered_multiset = tree<T,
8     null_type, less_equal<T>, rb_tree_tag,
9     tree_order_statistics_node_update>;
10 template <typename T> using o_set = tree<T, null_type
11     , less<T>, rb_tree_tag,
12     tree_order_statistics_node_update>;
13 template <typename T, typename R> using o_map = tree<
14     T, R, less<T>, rb_tree_tag,
15     tree_order_statistics_node_update>;
16
17 int main() {
18     int i, j, k, n, m;
19     o_set<int> st;
20     st.insert(1);
21     st.insert(2);
22     cout << *st.find_by_order(0) << endl; /// k-esimo
23     elemento
24     cout << st.order_of_key(2) << endl; ///numero de
25     elementos menores que k
26     o_map<int, int> mp;
27     mp.insert({1, 10});
28     mp.insert({2, 20});
29     cout << mp.find_by_order(0)->second << endl; /// k-
30     esimo elemento

```

```

22     cout << mp.order_of_key(2) << endl; /// numero de
23     elementos (chave) menores que k
24     return 0;
25 }

```

## 6.2 Dsu

```

1 struct DSU {
2     vector<int> par, rank, sz;
3     int c;
4     DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n +
5     1, 1), c(n) {
6         for (int i = 1; i <= n; ++i) par[i] = i;
7     }
8     int find(int i) {
9         return (par[i] == i ? i : (par[i] = find(par[
10        i])));
11    }
12    bool same(int i, int j) {
13        return find(i) == find(j);
14    }
15    int get_size(int i) {
16        return sz[find(i)];
17    }
18    int count() {
19        return c; // quantos componentes conexos
20    }
21    int merge(int i, int j) {
22        if ((i = find(i)) == (j = find(j))) return
23        -1;
24        else --c;
25        if (rank[i] > rank[j]) swap(i, j);
26        par[i] = j;
27        sz[j] += sz[i];
28        if (rank[i] == rank[j]) rank[j]++;
29        return j;
30    }
31 };

```

## 7 Primitives

## 8 General

### 8.1 Bitwise

```

1 int check_kth_bit(int x, int k) {
2     return (x >> k) & 1;
3 }
4
5 void print_on_bits(int x) {
6     for (int k = 0; k < 32; k++) {
7         if (check_kth_bit(x, k)) {
8             cout << k << ' ';
9         }
10    }
11    cout << '\n';
12 }
13
14 int count_on_bits(int x) {
15     int ans = 0;
16     for (int k = 0; k < 32; k++) {
17         if (check_kth_bit(x, k)) {
18             ans++;
19         }
20    }
21    return ans;
22 }
23
24 bool is_even(int x) {
25     return ((x & 1) == 0);
26 }
27
28 int set_kth_bit(int x, int k) {
29     return x | (1 << k);
30 }
31
32 int unset_kth_bit(int x, int k) {
33     return x & ~(1 << k);
34 }
35
36 int toggle_kth_bit(int x, int k) {
37     return x ^ (1 << k);
38 }
39
40 bool check_power_of_2(int x) {
41     return count_on_bits(x) == 1;
42 }

```