

Competitive Programming Notebook

Programadores Roblox

Contents

1	DP	2
1.1	Lis	2
1.2	Lcs	2
1.3	Knapsack	2
2	String	2
3	Geometry	2
4	Graph	2
4.1	Dijkstra	2
5	Math	2
5.1	Fexp	2
6	DS	2
7	Primitives	2
8	General	2
8.1	Bitwise	2

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;

```

5 Math

5.1 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 }

```

6 DS

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 }

```