

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 Kruskal	2
4.2 Topological Sort	2
4.3 Floyd Warshall	2
4.4 Dijkstra	3
5 Math	3
5.1 Crivo	3
5.2 Exgcd	3
5.3 Fexp	3
5.4 Equacao Diofantina	3
6 DS	3
6.1 Ordered Set E Map	3
6.2 Dsu	3
7 Primitives	4
8 General	4
8.1 Bitwise	4

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 Kruskal

```

1 // Ordena as arestas por peso, insere se ja nao
  estiver no mesmo componente
2 // O(E log E)
3
4 struct DSU {
5     vector<int> par, rank, sz;
6     int c;
7     DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n +
8 1, 1), c(n) {
9         for (int i = 1; i <= n; ++i) par[i] = i;
10    }
11    int find(int i) {
12        return (par[i] == i ? i : (par[i] = find(par[
13 i])));
14    }
15    bool same(int i, int j) {
16        return find(i) == find(j);
17    }
18    int get_size(int i) {
19        return sz[find(i)];
20    }
21    int count() {
22        return c; // quantos componentes conexos
23    }
24    int merge(int i, int j) {
25        if ((i = find(i)) == (j = find(j))) return
26 -1;
27        else --c;
28        if (rank[i] > rank[j]) swap(i, j);
29        par[i] = j;
30        sz[j] += sz[i];

```

```

28         if (rank[i] == rank[j]) rank[j]++;
29         return j;
30     }
31 };
32
33 struct Edge {
34     int u, v, w;
35     bool operator <(Edge const & other) {
36         return weight < other.weight;
37     }
38 }
39
40 vector<Edge> kruskal(int n, vector<Edge> edges) {
41     vector<Edge> mst;
42     DSU dsu = DSU(n + 1);
43     sort(edges.begin(), edges.end());
44     for (Edge e : edges) {
45         if (dsu.find(e.u) != dsu.find(e.v)) {
46             mst.push_back(e);
47             dsu.join(e.u, e.v);
48         }
49     }
50     return mst;
51 }

```

4.2 Topological Sort

```

1 vector<int> adj[MAXN];
2 vector<int> estado(MAXN); // 0: nao visitado 1:
  processamento 2: processado
3 vector<int> ordem;
4 bool temCiclo = false;
5
6 void dfs(int v) {
7     if(estado[v] == 1) {
8         temCiclo = true;
9         return;
10    }
11    if(estado[v] == 2) return;
12    estado[v] = 1;
13    for(auto &nei : adj[v]) {
14        if(estado[v] != 2) dfs(nei);
15    }
16    estado[v] = 2;
17    ordem.push_back(v);
18    return;

```

4.3 Floyd Warshall

```

1 // SSP e acha ciclos.
2 // Bom com constraints menores.
3 // O(n^3)
4
5 int dist[501][501];
6
7 void floydWarshall() {
8     for(int k = 0; k < n; k++) {
9         for(int i = 0; i < n; i++) {
10            for(int j = 0; j < n; j++) {
11                dist[i][j] = min(dist[i][j], dist[i][
12 k] + dist[k][j]);
13            }
14        }
15    }
16
17 void solve() {
18     int m, q;
19     cin >> n >> m >> q;
20     for(int i = 0; i < n; i++) {
21         for(int j = i; j < n; j++) {
22             if(i == j) {
23                 dist[i][j] = dist[j][i] = 0;

```

```

23         } else {
24             dist[i][j] = dist[j][i] = linf;
25         }
26     }
27 }
28 for(int i = 0; i < m; i++) {
29     int u, v, w;
30     cin >> u >> v >> w; u--; v--;
31     dist[u][v] = min(dist[u][v], w);
32     dist[v][u] = min(dist[v][u], w);
33 }
34 floydWarshall();
35 while(q--){
36     int u, v;
37     cin >> u >> v; u--; v--;
38     if(dist[u][v] == linf) cout << -1 << '\n';
39     else cout << dist[u][v] << '\n';
40 }
41 }

```

4.4 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 Crivo

```

1 // O(n*log(log(n)))
2 bool composto[MAX]
3 for(int i = 1; i <= n; i++) {
4     if(composto[i]) continue;
5     for(int j = 2*i; j <= n; j += i)
6         composto[j] = 1;
7 }

```

5.2 Exgcd

```

1 // O 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 }

```

5.3 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 }

```

5.4 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 Ordered Set E Map

```

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
31     cout << mp.order_of_key(2) << endl; /// numero de
32     elementos (chave) menores que k
33     return 0;
34 }

```

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) {

```

```

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

```

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 }

```