

# Lab2 Lib

Pedro Vidal

## 1. Defines

```
1 #define int long long
2 #define INF LLONG_MAX
3 #define _INF LLONG_MIN
4 #define ta a.size() + 1
5 #define tb b.size() + 1
6 #define vi vector<int>
7 #define vvi vector<vi>
8 #define pii pair<int, int>
9 #define endl '\n'
```

## 2. DP

### 2.1. Prob da Moeda

```
1 vi dp(m + 1, INF), coins(n), ultima(m + 1, 0); dp[0] = 0;
2 // read and sort coins
3 // bitset<100001> dp; dp[0] = true // solucao bitset
4 for (int i = 0; i < n; i++){
5     // dp |= (dp << coin[i]); // solucao bitset
6     for (int j = 0; j <= m && j + coins[i] <= m; j++){ // moedas ilimitadas
7         // for (int j = m; j >= 0; j--){ //moedas limitadas
8             if (dp[j] + 1 < dp[j + moeda[i]]) ultima[j + moeda[i]] = moeda[i];
9             dp[j + coins[i]] = min(dp[j + coins[i]], dp[j] + 1);
10        }
11    }
12    dp[m] == INF ? cout << "Impossivel" : cout << dp[m]; cout << endl;
```

### 2.2. Prob da Mochila

```
1 vi p(n), v(n), dp(cap+1, _INF); dp[0] = 0;
2 // read weight (p) and value (v)
3 int res = 0;
4 for (int i = 0; i < n; i++){
5     for (int j = cap; j >= 0; j--){
6         if (dp[j] == _INF || j + p[i] > cap) continue;
7         dp[j + p[i]] = max(dp[j + p[i]], dp[j] + v[i]);
8         res = max(res, dp[j + p[i]]);
9     }
10 }
```

### 2.3. Longest Common Subsequence (LCS)

```
1 vvi lcs(ta, vi(tb, 0));
2 for (int i = 1; i < ta; i++){
3     for (int j = 1; j < tb; j++){
4         lcs[i][j] = (a[i - 1] == b[j - 1] ? lcs[i - 1][j - 1] + 1 :
5             max(lcs[i - 1][j], lcs[i][j - 1]));
6     }
7     cout << lcs[ta - 1][tb - 1] << endl;
```

### 2.4. Bactracking (LCS)

```
1 string res;
2 int i = ta - 1, j = tb - 1;
3 while(i + j){
4     if (a[i - 1] == b[j - 1]){
5         res += a[i - 1];
6         i--; j--;
7     }
8     else if (j == 0 || (i && lcs[i - 1][j] > lcs[i][j - 1])){
9         // res += a[i - 1]; // menor string q contenha a e b como subseq
10        i--;
11    }
12    else if (j){
13        // res += b[j - 1]; // menor string q contenha a e b como subseq
14        j--;
15    }
16 }
17 reverse(res.begin(), res.end());
18 cout << res << endl;
```

### 2.5. Edit Distance

```
1 vvi edist(ta, vi(tb, 0));
2 for (int i = 0; i < ta; i++){
3     for (int j = 0; j < tb; j++){
4         if (i == 0 || j == 0){ edist[i][j] = max(i, j); continue;}
5         edist[i][j] = (a[i - 1] == b[j - 1] ? edist[i - 1][j - 1] : min({edist[i - 1][j - 1],
6             edist[i - 1][j], edist[i][j - 1]}) + 1);
7     }
8 }
9 cout << edist[ta - 1][tb - 1] << endl;
```

### 2.6. Longest Increasing Subsequence (LIS)

```
1 vi lis(v.size() + 1, INF);
2 lis[0] = _INF;
3 int res = 0;
4 for (int i = 0; i < v.size(); i++){
5     int pos = lower_bound(lis.begin(), lis.end(), v[i]) - lis.begin();
6     lis[pos] = v[i];
7     res = max(res, pos);
8 }
```

### 2.7. Using LIS to solve LCS

```
1 int calc_lis(vi & vet, int n){
2     vi lis(n + 1, INF);
3     lis[0] = 0;
4     int res = 0;
5     for (int i = 0; i < n; i++){
6         int pos = lower_bound(lis.begin(), lis.end(), vet[i]) - lis.begin();
```

```

7     lis[pos] = vet[i];
8     res = max(res, pos);
9 }
10 return res;
11 }
12
13 signed main(){
14     vi vet(n);
15     map<int, int> pos_id;
16     for (int i = 0; i < n; i++){
17         cin >> vet[i];
18         pos_id[vet[i]] = i + 1;
19     }
20     for (int i = 0; i < n; i++){
21         cin >> vet[i];
22         vet[i] = pos_id[vet[i]];
23     }
24     cout << calc_lis(vet, n) << endl;
25 }

```

### 3. Grafos

#### 3.1. SCC

```

1 void dfs(int u, vvi & adj, vector<bool> & vis, vi & vet){
2     vis[u] = true;
3     for (int v : adj[u]){
4         if (!vis[v]){
5             dfs(v, adj, vis, vet);
6         }
7     }
8     vet.push_back(u);
9 }
10
11 signed main(){
12     vvi adj(n), adjT(n), scc;
13     vector<bool> vis(n, false);
14     vi ordem_vis;
15     // read edges and fill adj & adjT
16     for (int i = 0; i < n; i++){
17         if (!vis[i]){ dfs(i, adj, vis, ordem_vis); }
18     }
19     vis.assign(n, false);
20     for (int i = n - 1; i >= 0; i--){
21         if (!vis[ordem_vis[i]]){
22             scc.push_back(vi());
23             dfs(ordem_vis[i], adjT, vis, scc[scc.size()-1]);
24         }
25     }
26 }

```

#### 3.2. Bridges

```

1 void dfs(int u, vvi & adj, vector<bool> & vis, vector<pii> & res, int & tempo,
2         vi & low, vi & d, vi & pai){
3     tempo++;
4     d[u] = low[u] = tempo;
5     vis[u] = true;
6     for (int v : adj[u]){
7         if (!vis[v]){
8             pai[v] = u;
9             dfs(v, adj, vis, res, tempo, low, d, pai);
10            if (low[v] > d[u]){
11                res.emplace_back(min(u, v), max(u, v));
12            }
13            low[u] = min(low[u], low[v]);
14        }
15        else if (pai[u] != v){
16            low[u] = min(low[u], low[v]);
17        }
18    }
19 }

```

```

13     }
14     else if (pai[u] != v){
15         low[u] = min(low[u], d[v]);
16     }
17 }
18 }
19
20 signed main(){
21     vvi adj(n);
22     vector<bool> vis(n, false);
23     vector<pii> res;
24     vi low(n, INF), d(n, INF), pai(n, INF);
25     // read edges and fill adj
26     int tempo = 0;
27     for (int i = 0; i < n; i++){
28         if (!vis[i]){ dfs(i, adj, vis, res, tempo, low, d, pai); }
29     }
30 }

```

#### 3.3. Pontos de Articulação

```

1 struct Grafo{
2     vvi adj;
3     vi low, d, pai;
4     vector<bool> vis;
5     set<int> articulations;
6     int tempo = 0;
7
8     Grafo(int n){
9         adj.resize(n);
10        low.resize(n, INF);
11        d.resize(n, INF);
12        pai.resize(n, INF);
13        vis.resize(n, false);
14    }
15 };
16
17 int dfs(int u, Grafo & g, bool root){
18     g.tempo++;
19     g.d[u] = g.low[u] = g.tempo;
20     g.vis[u] = true;
21     int cont = 0;
22     for (int v : g.adj[u]){
23         if (!g.vis[v]){
24             cont++;
25             g.pai[v] = u;
26             dfs(v, g, false);
27             if (!root && g.low[v] >= g.d[u]){
28                 g.articulations.insert(u);
29             }
30             g.low[u] = min(g.low[u], g.low[v]);
31         }
32         else if (g.pai[u] != v){
33             g.low[u] = min(g.low[u], g.d[v]);
34         }
35     }
36     return cont;
37 }
38
39 signed main(){
40     Grafo g = Grafo(n);
41     // read edges and fill g.adj
42     for (int i = 0; i < n; i++){
43         if (!g.vis && dfs(i, g, true) > 1){
44             g.articulations.insert(i);
45         }
46     }

```

47

## 3.4. MST

```

1 struct edge{ int x, y, w; };
2
3 int find(int x, vi & uf){ return uf[x] == x ? x : find(uf[x], uf); }
4
5 void merge(int x, int y, vi & uf, vi & sz){
6     x = find(x, uf); y = find(y, uf);
7     if (sz[x] > sz[y]){ swap(x, y); }
8     sz[y] += sz[x];
9     uf[x] = uf[y];
10 }
11
12 signed main(){
13     vector<edge> vet(m);
14     // read edges and sort by weight
15     vi uf(n), sz(n, 1);
16     iota(uf.begin(), uf.end(), 0);
17     int res = 0;
18     for (int i = 0; i < m; i++){
19         if (find(vet[i].x, uf) != find(vet[i].y, uf)){
20             merge(vet[i].x, vet[i].y, uf, sz);
21             res += vet[i].w;
22         }
23     }
24 }

```

## 3.5. Segunda MST

```

1 struct Edge{ int u, v, w; bool usable = true; };
2
3 int find(vi & uf, int x){ return uf[x] == x ? x : find(uf, uf[x]); }
4
5 void merge(vi & uf, vi & sz, int x, int y){
6     x = find(uf, x); y = find(uf, y);
7     if (sz[x] > sz[y]){ swap(x, y); }
8     sz[y] += sz[x];
9     uf[x] = uf[y];
10 }
11
12 pair<vi, int> mst(vector<Edge> & edges, vi uf, vi sz, int n){
13     vi ids;
14     int m = edges.size(), cost = 0;
15     for (int i = 0; i < m; i++){
16         if (edges[i].usable && find(uf, edges[i].u) != find(uf, edges[i].v)){
17             merge(uf, sz, edges[i].u, edges[i].v);
18             ids.push_back(i);
19             cost += edges[i].w;
20         }
21     }
22     return ids.size() == n - 1 ? make_pair(ids, cost) : make_pair(vi (), INF);
23 }
24
25 signed main(){
26     vi sz(n + 1, 1), uf(n + 1), mst_orig, dummy;
27     iota(uf.begin(), uf.end(), 0);
28     vector<Edge> edges(m);
29     // read edges and sort by weight
30     int cost_orig, second_best = INF, cost;
31     tie(mst_orig, cost_orig) = mst(edges, uf, sz, n);
32     if (cost_orig == INF){ cout << "No way" << endl; return 0; }
33     for (int id : mst_orig){
34         edges[id].usable = false;
35         iota(uf.begin(), uf.end(), 0);

```

```

36     sz.assign(n + 1, 1);
37     tie(dummy, cost) = mst(edges, uf, sz, n);
38     second_best = min(second_best, cost);
39     edges[id].usable = true;
40 }
41     second_best == INF ? cout << "No second way" : cout << second_best; cout <<
42     endl;
43 }

```

## 3.6. Dijkstra

```

1 struct Grafo{ vvi adj(n), cost(n); vi dist(n, INF); };
2
3 int dijkstra(int s, int t, Grafo g){
4     g.dist[s] = 0;
5     priority_queue<pii, vector<pii>, greater<pii>> pq;
6     pq.emplace(g.dist[s], s);
7     while(!pq.empty()){
8         int d, u;
9         tie(d, u) = pq.top(); pq.pop();
10        for (int i = 0; i < g.adj[u].size(); i++){
11            int v = g.adj[u][i], w = g.cost[u][i];
12            if (d + w < g.dist[v]){
13                g.dist[v] = d + w;
14                pq.emplace(g.dist[v], v);
15            }
16        }
17    }
18    return g.dist[t] == INF ? -1 : g.dist[t];
19 }

```

## 3.7. Bellman-Ford

```

1 void bellman_ford(vector<tuple<int, int, int>> edges, int n, int s){
2     vector<int> dist(n, INF);
3     dist[s] = 0;
4     for (int i = 0; i < n; i++){
5         for (int j = 0; j < edges.size(); j++){
6             int u = get<0>(edges[j]), v = get<1>(edges[j]), w = get<2>(edges[j]);
7             dist[v] = min(dist[v], dist[u] + w);
8         }
9     }
10    bool has_negative_cycle = false;
11    for (int j = 0; j < edges.size(); j++){
12        int u = get<0>(edges[j]), v = get<1>(edges[j]), w = get<2>(edges[j]);
13        if (dist[v] > dist[u] + w){
14            has_negative_cycle = true;
15        }
16    }
17 }

```