Lab2 Lib

Pedro Vidal

1. Defines

```cpp
#define int long long
#define INF LLONG_MAX
#define _INF LLONG_MIN
#define ta a.size() + 1
#define tb b.size() + 1
#define vi vector<int>
#define vvi vector<vi>
#define pii pair<int, int>
#define endl '\n'
```

2. DP

2.1. Prob da Moeda

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

2.2. Prob da Mochila

```cpp
vi p(n), v(n), dp(cap+1, _INF); dp[0] = 0;
// read weight (p) and value (v)
int res = 0;
for (int i = 0; i < n; i++){
  for (int j = cap; j >= 0; j--){
    if (dp[j] == _INF || j + p[i] > cap) continue;
    dp[j + p[i]] = max(dp[j + p[i]], dp[j] + v[i]);
    res = max(res, dp[j + p[i]]);
  }
}
```

2.3. Longest Common Subsequence (LCS)

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

2.4. Edit Distance

```cpp
vvi edist(ta, vi(tb, 0));
for (int i = 0; i < ta; i++){
  for (int j = 0; j < tb; j++){
    if (i == 0 || j == 0){ edist[i][j] = max(i, j); continue;}
    edist[i][j] = (a[i-1] == b[j-1] ? edist[i-1][j-1] : min({edist[i-1][j-1],
    edist[i-1][j], edist[i][j-1]}) + 1);
  }
}
cout << edist[ta-1][tb-1] << endl;
```

2.5. Longest Increasing Subsequence (LIS)

```cpp
vi lis(v.size() + 1, INF);
lis[0] = _INF;
int res = 0;
for (int i = 0; i < v.size(); i++){
  int pos = lower_bound(lis.begin(), lis.end(), v[i]) - lis.begin();
  lis[pos] = v[i];
  res = max(res, pos);
}
```

2.6. Using LIS to solve LCS

```cpp
int calc_lis(vi & vet, int n){
  vi lis(n + 1, INF);
  lis[0] = 0;
  int res = 0;
  for (int i = 0; i < n; i++){
    int pos = lower_bound(lis.begin(), lis.end(), vet[i])  - lis.begin();
    lis[pos] = vet[i];
    res = max(res, pos);
  }
  return res;
}

signed main(){
  vi vet(n);
  map<int, int> pos_id;
  for (int i = 0; i < n; i++){
    cin >> vet[i];
    pos_id[vet[i]] = i + 1;
  }
  for (int i = 0; i < n; i++){
    cin >> vet[i];
    vet[i] = pos_id[vet[i]];
  }
  cout << calc_lis(vet, n) << endl;
}
```

3. Grafos

3.1. SCC

```cpp
void dfs(int u, vvi & adj, vector<bool> & vis, vi & vet){
  vis[u] = true;
  for (int v : adj[u]){
    if (!vis[v]){
      dfs(v, adj, vis, vet);
    }
  }
  vet.push_back(u);
}

signed main(){
  vvi adj(n), adjT(n), scc;
  vector<bool> vis(n, false);
  vi ordem_vis;
  // read edges and fill adj & adjT
  for (int i = 0; i < n; i++){
    if (!vis[i]){ dfs(i, adj, vis, ordem_vis); }
  }
  vis.assign(n, false);
  for (int i = n - 1; i >= 0; i--){
    if (!vis[ordem_vis[i]]){
      scc.push_back(vi());
      dfs(ordem_vis[i], adjT, vis, scc[scc.size()-1]);
    }
  }
}
```

3.2. Bridges

```cpp
void dfs(int u, vvi & adj, vector<bool> & vis, vector<pii> & res, int & tempo,
    vi & low, vi & d, vi & pai){
  tempo++;
  d[u] = low[u] = tempo;
  vis[u] = true;
  for (int v : adj[u]){
    if (!vis[v]){
      pai[v] = u;
      dfs(v, adj, vis, res, tempo, low, d, pai);
      if (low[v] > d[u]){
        res.emplace_back(min(u, v), max(u, v));
      }
      low[u] = min(low[u], low[v]);
    }
    else if (pai[u] != v){
      low[u] = min(low[u], d[v]);
    }
  }
}

signed main(){
  vvi adj(n);
  vector<bool> vis(n, false);
  vector<pii> res;
  vi low(n, INF), d(n, INF), pai(n, INF);
  // read edges and fill adj
  int tempo = 0;
  for (int i = 0; i < n; i++){
    if (!vis[i]){ dfs(i, adj, vis, res, tempo, low, d, pai); }
  }
}
```

3.3. Pontos de Articulação

```cpp
struct Grafo{
  vvi adj;
  vi low, d, pai;
  vector<bool> vis;
  set<int> articulations;
  int tempo = 0;

  Grafo(int n){
    adj.resize(n);
    low.resize(n, INF);
    d.resize(n, INF);
    pai.resize(n, INF);
    vis.resize(n, false);
  }
};

int dfs(int u, Grafo & g, bool root){
  g.tempo++;
  g.d[u] = g.low[u] = g.tempo;
  g.vis[u] = true;
  int cont = 0;
  for (int v : g.adj[u]){
    if (!g.vis[v]){
      cont++;
      g.pai[v] = u;
      dfs(v, g, false);
      if (!root && g.low[v] >= g.d[u]){
        g.articulations.insert(u);
      }
      g.low[u] = min(g.low[u], g.low[v]);
    }
    else if (g.pai[u] != v){
      g.low[u] = min(g.low[u], g.d[v]);
    }
  }
  return cont;
}

signed main(){
  Grafo g = Grafo(n);
  // read edges and fill g.adj
  for (int i = 0; i < n; i++){
    if (!g.vis && dfs(i, g, true) > 1){
      g.articulations.insert(i);
    }
  }
}
```

3.4. MST

```cpp
struct edge{ int x, y, w; };

int find(int x, vi & uf){ return uf[x] == x ? x : find(uf[x], uf); }

void merge(int x, int y, vi & uf, vi & sz){
  x = find(x, uf); y = find(y, uf);
  if (sz[x] > sz[y]){ swap(x, y); }
  sz[y] += sz[x];
  uf[x] = uf[y];
}

signed main(){
  vector<edge> vet(m);
  // read edges and sort by weight
  vi uf(n), sz(n, 1);
```

```
  iota(uf.begin(), uf.end(), 0);
  int res = 0;
  for (int i = 0; i < m; i++){
    if (find(vet[i].x, uf) != find(vet[i].y, uf)){
      merge(vet[i].x, vet[i].y, uf, sz);
      res += vet[i].w;
    }
  }
}
```

3.5. Segunda MST

```
struct Edge{ int u, v, w; bool usable = true; };

int find(vi & uf, int x){ return uf[x] == x ? x : find(uf, uf[x]); }

void merge(vi & uf, vi & sz, int x, int y){
  x = find(uf, x); y = find(uf, y);
  if (sz[x] > sz[y]){ swap(x, y); }
  sz[y] += sz[x];
  uf[x] = uf[y];
}

pair<vi, int> mst(vector<Edge> & edges, vi uf, vi sz, int n){
  vi ids;
  int m = edges.size(), cost = 0;
  for (int i = 0; i < m; i++){
    if (edges[i].usable && find(uf, edges[i].u) != find(uf, edges[i].v)){
      merge(uf, sz, edges[i].u, edges[i].v);
      ids.push_back(i);
      cost += edges[i].w;
    }
  }
  return ids.size() == n - 1 ? make_pair(ids, cost) : make_pair(vi (), INF);
}

signed main(){
  vi sz(n + 1, 1), uf(n + 1), mst_orig, dummy;
  iota(uf.begin(), uf.end(), 0);
  vector<Edge> edges(m);
  // read edges and sort by weight
  int cost_orig, second_best = INF, cost;
  tie(mst_orig, cost_orig) = mst(edges, uf, sz, n);
  if (cost_orig == INF){ cout << "No way" << endl; return 0; }
  for (int id : mst_orig){
    edges[id].usable = false;
    iota(uf.begin(), uf.end(), 0);
    sz.assign(n + 1, 1);
    tie(dummy, cost) = mst(edges, uf, sz, n);
    second_best = min(second_best, cost);
    edges[id].usable = true;
  }
  second_best == INF ? cout << "No second way" : cout << second_best; cout <<
    endl;
}
```

3.6. Dijkstra

```
struct Grafo{ vvi adj(n), cost(n); vi dist(n, INF); };

int dijkstra(int s, int t, Grafo g){
  g.dist[s] = 0;
  priority_queue<pii, vector<pii>, greater<pii>> pq;
  pq.emplace(g.dist[s], s);
  while(!pq.empty()){
    int d, u;
    tie(d, u) = pq.top(); pq.pop();
    for (int i = 0; i < g.adj[u].size(); i++){
      int v = g.adj[u][i], w = g.cost[u][i];
      if (d + w < g.dist[v]){
        g.dist[v] = d + w;
        pq.emplace(g.dist[v], v);
      }
    }
  }
  return g.dist[t] == INF ? -1 : g.dist[t];
}
```

3.7. Bellman-Ford

```
void bellman_ford(vector<tuple<int, int, int>> edges, int n, int s){
  vector<int> dist(n, INF);
  dist[s] = 0;
  for (int i = 0; i < n; i++){
    for (int j = 0; j < edges.size(); j++){
      int u = get<0>(edges[j]), v = get<1>(edges[j]), w = get<2>(edges[j]);
      dist[v] = min(dist[v], dist[u] + w);
    }
  }
  bool has_negative_cycle = false;
  for (int j = 0; j < edges.size(); j++){
    int u = get<0>(edges[j]), v = get<1>(edges[j]), w = get<2>(edges[j]);
    if (dist[v] > dist[u] + w){
      has_negative_cycle = true;
    }
  }
}
```