

Aula prática 9

Esta aula tem como objetivo estudar a estrutura de dados “grafo”, explorando a implementação baseada em listas e matriz de adjacências e alguns dos algoritmos base.

- 1 Pretende-se implementar um **dígrafo** recorrendo ao uso de *listas de adjacências*. Um grafo com n vértices pode ser representado por uma lista de n vértices e seus respectivos vizinhos.

Considere a classe *Graph*. São fornecidos os ficheiros “.cpp” e “.hpp” com as estruturas definidas e código parcialmente implementado. O ficheiro **test_graph.cpp** contém a função `main()`, que pode ser utilizada para implementar testes às funções pedidas.

Estude cuidadosamente a estrutura de dados fornecida, observando que o *grafo* guarda o número total de vértices e listas de adjacências.

- a) Implemente as seguintes funções de adição, remoção e teste de existência de arestas:

`int addEdgeDirected(int v1, int v2)`

`int edgeRemove(int v1, int v2)`

`int existsNeighbour(int v1, int v2)`

As funções devem efetuar as verificações necessárias, incluindo se as variáveis *v1* e *v2* estão dentro dos intervalos esperados. Em caso de erro devem retornar -1.

Exemplo

Grafo criado com 5 nos e 7 arestas.

```
0 -> 2
1 -> 3
2 -> 1
3 -> 0 2 4
4 -> 0
```

Aresta removida com sucesso.

```
0 -> 2
1 -> 3
2 -> 1
3 -> 2 4
4 -> 0
```

Os nos 1 e 2 sao vizinhos
Os nos 0 e 1 nao sao vizinhos

- b) Implemente a função **outDegree**, que devolve a quantidade de vértices que sai de um determinado vértice.

`int outDegree (int v)`

Exemplo

Grau de saida do vertice 1: 1
Grau de saida do vertice 3: 2

- c) Implemente a função **transposeGraph** que gera um grafo transposto (com todas as arestas no sentido contrário).

vector<list<int>> transposeGraph(Graph* g)

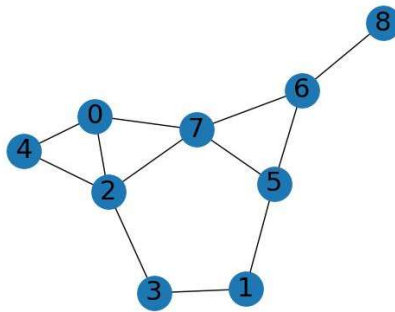
Exemplo

Grafo transposto:

```
0 -> 4
1 -> 2
2 -> 0 3
3 -> 1
4 -> 3
```

- 2 A rede do sistema de distribuição de energia é representada através de um grafo $G(V, E)$, onde V é o conjunto de subestações e E é o conjunto de linhas de transmissão. Para planejar uma expansão do sistema, a FEUP Energy precisa de identificar a subestação que está com sobrecarga. Tendo por base a biblioteca de grafos **grafo.cpp/.hpp**, implemente as funções pedidas no ficheiro grafo.cpp.

- a) Implemente a função **addEdgeUndirected** para criar o grafo **não direcionado**, que está representado na figura abaixo. A função deverá efetuar as verificações necessárias, incluindo se as variáveis **v1** e **v2** estão dentro dos intervalos esperados. Em caso de erro devem retornar -1.



Exemplo

Grafo criado com 9 nós e 12 arestas.

```
0 -> 4 2 7
1 -> 3 5
2 -> 0 3 4 7
3 -> 1 2
4 -> 0 2
5 -> 1 6 7
6 -> 5 7 8
7 -> 0 2 5 6
8 -> 6
```

- b) Implemente a função **int identifyOverload()**, que deverá encontrar a subestação com o maior número de linhas de transmissão. Caso encontre mais que uma subestação, deverá retornar a de maior valor.

Exemplo

Subestacao: 7

- 3 Considere a declaração e implementação da classe ***SocialNetwork*** nos ficheiros ***social_net.hpp*** e ***social_net.cpp***. Esta classe utiliza uma matriz de adjacências para implementar uma rede social em que os nodos, identificados por um número inteiro, representam as pessoas e as arestas as relações de amizade entre elas. Implemente no ficheiro ***social_net.cpp*** os seguintes métodos:

- a) ***SocialNetwork::addFriendship(int u, int v)*** que aceita como parâmetros dois identificadores de pessoas e cria uma ligação de amizade entre elas. O teste deste método através do ficheiro ***social_net_test.cpp*** deverá ter o seguinte resultado:

Exemplo
Social Network Graph (Adjacency Matrix): Person 0: 0 1 1 0 0 0 Person 1: 1 0 1 1 0 0 Person 2: 1 1 0 0 1 0 Person 3: 0 1 0 0 1 0 Person 4: 0 0 1 1 0 1 Person 5: 0 0 0 0 1 0

- b) ***SocialNetwork::findMutualFriends(int person1, int person2)*** que aceita como parâmetros dois identificadores de pessoas e imprime no ecrã os identificadores dos amigos comuns a essas pessoas. O teste deste método através do ficheiro ***social_net_test.cpp*** deverá ter o seguinte resultado:

Exemplo
Mutual friends between person 0 and person 1: 2 Mutual friends between person 2 and person 3: 1 4 Mutual friends between person 1 and person 4: 2 3

- c) ***SocialNetwork::findConnectedGroups()*** que imprime no ecrã os grupos fechados de amigos existentes numa rede: grupos em que nenhuma das pessoas é amiga de qualquer pessoa de outro grupo. Dica: analise e utilize o método ***SocialNetwork::DFSUtil(int v, vector<bool>& visited)*** que faz uma travessia do grafo do tipo *Depth-First Search*. O teste deste método através do ficheiro ***social_net_test.cpp*** deverá ter o seguinte resultado:

Exemplo
Connected group: 0 1 2 Connected group: 3 4 5