4ª Aula Prática – Grafos: Representação, pesquisa profundidade e largura, ordenação topológica

Instruções

- Faça download do ficheiro **cal_fp04_CLion.zip** da página da disciplina e descomprima-o (contém a pasta **lib,** a pasta **Tests** com os ficheiros **tests.cpp**, **Graph.h**, **Person.h**, **Person.cpp**, e os ficheiros **CMakeLists** e **main.cpp**)
- No CLion, abra um ***projeto***, selecionando a pasta que contém os ficheiros do ponto anterior.
- Efetuar "*Load CMake Project*" sobre o ficheiro *CMakeLists.txt*
- Execute o projeto (**Run**)
- Note que os ***testes unitários deste projeto podem estar comentados***. Se for este o caso, retire os comentários à medida que vai implementando os testes.
- ***Deverá realizar esta ficha respeitando a ordem das alíneas***.
- Efetue a implementação no ficheiro nos respetivos ficheiros ***.cpp***, no caso de não estar a implementar um template. Os templates deverão ser implementados nos próprios ficheiros ***.h***.
- Nota importante: se necessitar ler ficheiros de texto em modo I/O, deverá configurar a sua localização no CLion, redefinindo a variável do ambiente IDE "Working Directory", a parir do menu Run > Edit Configurations… > Working Direcory.
- O código a completar no ficheiro **Graph.h** está marcado com **TODO** e acompanhado de comentários explicativos e dicas.

Enunciado

1. Grafos: representação/manipulação

Considere a classe **Graph** definida no ficheiro *Graph.h:*

```
template <class T>  class Vertex {
	T info;
	vector<Edge<T> > adj;
public:
	…
	friend class Graph<T>;
};

template <class T>  class Edge {
	Vertex<T> * dest;
	double weight;
public:
	…
	friend class Graph<T>;
	friend class Vertex<T>;
};
```

```
template <class T> class Graph {
	vector<Vertex<T> *> vertexSet;
	…
public:
	…
};
```

a) Implemente na classe **Graph** o membro-função:
```
bool addVertex(const T &in)
```
Esta função adiciona o vértice com conteúdo ***in*** (info) ao grafo. Retorna ***true*** se o vértice foi adicionado com sucesso e ***false*** se já existe um com o mesmo conteúdo.

b) Implemente na classe **Graph** o membro-função:
```
bool addEdge(const T &sourc, const T &dest, double w)
```
Esta função adiciona ao grafo a aresta com origem no vértice ***sourc***, destino no vértice ***dest*** e peso ***w***. Retorna ***true*** se a aresta foi adicionado com sucesso e ***false*** se não é possível inserir essa aresta (por não existirem os vértices com o conteúdo indicado).

c) Implemente na classe **Graph** o membro-função:
```
bool removeEdge(const T &sourc, const T &dest)
```
Esta função remove a aresta com origem no nó ***sourc*** e destino no nó ***dest***. Retorna ***true*** se a aresta foi removida com sucesso e ***false*** senão (aresta não existe).

d) Implemente na classe **Graph** o membro-função:
```
bool removeVertex(const T &in)
```
Esta função remove o nó com o contéudo ***in***. Retorna ***true*** se o nó foi removido com sucesso e ***false*** no caso contrário (nó não existe). A remoção de um nó implica a remoção de todas as arestas com origem e/ou destino nesse nó.

2. Grafos: Pesquisa em profundidade e largura; ordenação topológica

a) Implemente na classe **Graph** o membro-função, de acordo com o algoritmo dados nas aulas teóricas:
```
vector<T> dfs() const
```
Esta função retorna um vetor contendo os elementos do grafo (conteúdo dos nós) quando sobre este é efectuada uma pesquisa em profundidade.

b) Implemente na classe **Graph** o membro-função, de acordo com o algoritmo dados nas aulas teóricas:
```
vector<T> bfs(Vertex<T> *s) const
```
Esta função retorna um vetor contendo os elementos do grafo (conteúdo dos nós) quando sobre este é efectuada uma pesquisa em largura a partir de **s**.

e) Implemente na classe **Graph** o membro-função, de acordo com o algoritmo dados nas aulas teóricas:

```
vector<T> topsort()
```

Esta função retorna um vetor contendo os elementos do grafo (membro-dado ***info*** dos vértices) ordenados topologicamente. Quando uma ordenação topológica não for possível para o grafo em questão, ou seja, quando não se tratar de um DAG, o vetor a retornar estará vazio.

3. Aplicações de pesquisa em largura e profundidade (PARA CASA)

a) Uma rede social é representada pela classe **Graph**, onde os nós representam indivíduos da rede e as arestas uma relação de amizade (o sentido da aresta indica quem enviou o pedido de amizade).

Após ter tomado conhecimento de uma notícia importante, um indivíduo divulga-a pelas suas amizades. Determine qual o indivíduo (nó) que divulga a notícia em primeira mão (como novidade) ao maior número de destinatários e qual o número de destinatários alcançados.

Implemente na classe **Graph** o membro-função:

```
int maxNewChildren(Vertex<T> *v, T &inf) const
```

Esta função retorna o número máximo de filhos de um nó do grafo (a partir de **v**); o parâmetro **inf** é o conteúdo desse nó do grafo. *Sugestão*: adapte o algoritmo de pesquisa em largura.

b) Implemente na classe **Graph** o membro-função:

```
bool isDAG()
```

Esta função verifica se o grafo dirigido é acíclico (*Directed Acyclic Graph*), isto é, não contém ciclos, caso em que a função retorna ***true***. Caso contrário, a função retorna ***false***. Sugestão: adapte o algoritmo de pesquisa em profundidade.