5ª Aula Prática – Grafos: Caminho mais curto

Instruções

- Faça download do ficheiro **cal_fp05_CLion.zip** da página da disciplina e descomprima-o (contém a pasta **lib,** a pasta **Tests** com os ficheiros **tests.cpp**, **Graph.h**, 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 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

Considere a classe **Graph** definida no ficheiro *Graph.h* e já utilizada nas aulas anteriores. Deverá atualizar as classes do ficheiro *Graph.h* adequadamente, a fim de realizar as alíneas que se seguem. Identifique a partir do ficheiro Test.cpp funções auxiliares que sejam necessárias e não sejam pedidas explicitamente nos exercícios.

1. Algoritmo de caminho mais curto em grafos não pesados

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

```
void unweightedShortestPath(const T &origin)
```

Esta função implementa um algoritmo para encontrar os caminhos mais curtos a partir do elemento ***v*** do grafo (vértice cujo conteúdo é ***origin***) a todos os outros vértices do grafo, ignorando os pesos das arestas.

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

```
vector<T> getPathTo(const T &dest)
```

Considerando que a propriedade ***path*** dos vértices do grafo foi atualizada pela invocação de um algoritmo de caminho mais curto de um vértice origem a todos os outros, esta função retorna um vetor

com a sequência dos vértices do caminho, desde a origem até ***dest***, inclusive (***dest*** é o membro-dado ***info*** do vértice de destino do caminho). Pressupõe-se que uma a função de cálculo de caminho, como **`unweightedShortestPath`**, foi chamada previamente com argumento ***origin***, que é o vértice de origem.

2. Algoritmo de Dijkstra

a) Implemente na classe **Graph** o membro-função público:

```
void dijkstraShortestPath(const T &origin)
```

Esta função implementa o algoritmo de Dijkstra para encontrar os caminhos mais curtos a partir de um vértice de origem (vértice ***s*** cujo conteúdo é ***origin***) para todos os outros vértices (ver algoritmo das aulas teóricas). Precisa de adicionar à classe *Vertex* campos para representar a distância mínima (***dist***) e o vértice anterior no caminho mais curto (***path***) (campos já criados no código fornecido).

Sugestão: Uma vez que a STL não disponibiliza filas de prioridade mutáveis (suportando *decrease_key*), usar uma fila de prioridades mutável de Boost ou usar a classe ***MutablePriorityQueue*** fornecida, que pode ser manipulada da seguinte forma:

- Para criar fila: `MutablePriorityQueue<Vertex<T> > q;`
- Para inserir elemento v (apontador para vértice): `q.insert(v);`
- Para extrair o mínimo (apontador para vértice): `v = q.extractMin();`
- Para avisar que chave (*dist*) de elemento *v* diminui de valor: `q.decreaseKey(v);`

Na classe ***Vextex*** é necessário (passos já realizados no código fornecido):

- Declarar campo `int queueIndex;`
- Declarar `friend class MutablePriorityQueue<Vertex<T> >;`
- Implementar `bool operator<(Vertex<T> & vertex) const` com base na comparação de valores do campo ***dist***.

b) Com base nos dados de desempenho do algoritmo de Dijkstra produzidos pelos testes fornecidos, crie um gráfico para mostrar que o tempo médio de execução é proporcional a $(|V| + |E|) \log_2 |V|$. Os testes de desempenho geram grafos aleatórios em forma de grelha de tamanho N x N, em que nº de vértices é $|V| = N^2$ e o nº de arestas é $4N(N-1)$.

PARA PRÓXIMA AULA

2. Outros algoritmos de caminho mais curto de um vértice para todos os outros.

a) Implemente na classe **Graph** o membro-função público:

```
void bellmanFordShortestPath(const T &origin)
```

Esta função implementa o algoritmo de Bellman-Ford para encontrar os caminhos mais curtos a partir do elemento *s* do grafo (vértice cujo conteúdo é ***origin***) a todos os outros vértices, permitindo a existência de arestas com pesos negativos.

3. Encontrar o caminho mais curto entre todos os pares de vértices.

a) Implemente na classe **Graph** o membro-função público:

```
void floydWarshallShortestPath()
```

Esta função implementa o algoritmo de Floyd-Warshall para encontrar os caminhos mais curtos entre todos os vértices ***v*** do grafo, no caso de grafos pesados (ver algoritmo e estruturas de dados nos slides das aulas teóricas). É necessário adicionar à classe **Graph** as matrizes referidas nos slides.

Adicionalmente, implemente na classe **Graph** o membro-função público:

```
vector<T> getfloydWarshallPath(const T &origin, const T &dest)
```

Esta função retorna um vetor com a sequência dos elementos do grafo representando os vértices do caminho de ***origin*** até ***dest***, inclusivé (onde ***origin*** e ***dest*** são as propriedades ***info*** dos vértices de origem e destino do caminho, respetivamente). Assuma que esta função é chamada depois da anterior.