6th Practical Class – Graphs: Shortest path

Instructions

- Download the zipped file **TP6_unsolved.zip** from the course's Moodle area and unzip it. It contains a cpp for each exercise, each with the respective unit tests, and the file **Graph.h** (based on the previous class).
- In the CLion IDE, open the project used in the previous lessons and add the folder TP6, selecting the folder that contains the files mentioned in the previous bullet point.
- Update the *CMakeLists.txt* file by copying, pasting, and adapting the three lines of code of TP6: file, add_executable and target_link_libraries.
- Do "*Load CMake Project*" over the file *CMakeLists.txt*
- Run the project (**Run**)
- Please note that the unit tests of this project may be commented. If this is the case, uncomment the tests as you make progress in the implementation of the respective exercises.
- You should implement the exercises following the order suggested.
- Implement your solutions in the respective .cpp file of each exercise.
- Important note: in case you need to read text files in I/O mode, you should tell CLion where such files are, by redefining the IDE environment variable "Working Directory", through menu Run > Edit Configurations… > Working Directory.

Exercises

1. Shortest path in unweighted graphs

a) Implement the following public method in the **Graph** class:

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

This method implements an algorithm to find the shortest paths from ***v*** (vertex which contains element *origin*) to all other vertices, ignoring edge weights.

b) Implement the following public member function in class **Graph**:

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

Considering that the ***path*** property of the graph's vertices has been updated by invoking a shortest path algorithm from one vertex *origin* to all others, this function returns a vector with the sequence of the vertices of the path, from the *origin* to ***dest***, inclusively (***dest*** is the attribute ***info*** of the destination vertex of the path). It is assumed that a path calculation function, such as `unweightedShortestPath`, was previously called with the *origin* argument, which is the origin vertex.

2. Dijkstra's algorithm

Consider the **Graph** class you used in previous classes, which is defined in the *Graph.h* file. You should edit the classes in *Graph.h* in order to complete the exercises below. Look at the *Test.cpp* file in order to identify auxiliary functions which are required but are not explicitly asked for.

a) Implement the following public member function in class **Graph**:

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

This method implements the Dijkstra algorithm to find the shortest paths from ***s*** (vertex which contains element ***origin***) to all other vertices, in a given weighted graph (see theoretical class slides). Update the **Vertex** class with member variables `int dist` and `Vertex* path`, representing the distance to the start vertex and the previous vertex in the shortest path, respectively. Since the STL doesn't support mutable priority queues, you can use the class provided ***MutablePriorityQueue***, as follows:

- To create a queue: `MutablePriorityQueue<Vertex<T> > q;`
- To insert vertex pointer ***v***: `q.insert(v);`
- To extract the element with minimum value (*dist*): v = `q.extractMin();`
- To notify that thee key (*dist*) of ***v*** was decreased:: `q.decreaseKey(v);`

b) Based on the performance data of the Dijkstra algorithm produced by the tests provided, create a chart to show that the average execution time is proportional to $(|V| + |E|) \log_2|V|$. The performance tests generate random graphs in the form of a grid of size N x N, in which the number of vertices is $| V | = N^2$ and the number of edges is 4N (N-1).

3. Other single source shortest path algorithms

a) Implement the following public method in **Graph** class:

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

This method implements the Bellman-Ford algorithm to find the shortest paths from ***v*** (vertex which contains element ***origin***) to all other vertices, in a given weighted graph.

4. All pairs shortest paths

a) Implement the following public method in the **Graph** class:

```
void floydWarshallShortestPath()
```

This method implements the Floyd-Warshall algorithm to find the shortest paths between all pairs of vertices in the graph.

Additionally, you will also have to implement the following public method of the **Graph** class:

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

This method returns a vector with the sequence of elements in the graph in the path from ***orgin*** to ***dest*** (where ***origin*** and ***dest*** are the values of the ***info*** member of the origin and destination vertices, respectively).