lab 22 Graphs via Adjacency Matrix

Instructions: In this lab implement a Graph with an adjacency matrix.

Implement the following class:

```
#ifndef GRAPHAM_H
#define GRAPHAM_H

/* This class represents a weighted directed graph via an adjacency matrix.
 * Vertices are given an index, starting from 0 and ascending
 * Class W : W represent the weight that can be associated with an edge.
 * We will not weight the vertices.
 * W is the data type for the weight. Normally an int.
 */

template<class W>
class GraphAM {
    private:
        /* Recommended, but not necessary. */
        void depthFirstTraversal(void (*visit)(const int node),
                int *visited, const int cVertex);
        /* You fill out private member variables. */
    public:
        /* Initialize an empty graph. */
        GraphAM();

        /* Initialize the Graph with a fixed number of vertices. */
        GraphAM(const int vertices);

        /* Deconstructor shall free up memory */
        ~GraphAM();

        /* Removes a vertex.
         * return whether successful or not
         * Note: You must shift all vertices accordingly.
         */
        bool removeVertex(int idx);

        /* Adds amt vertices to the graph. Returns the starting point
         * of the vertice count.
         */
        int addVertices(int amt);

        /* Adds an edge with weight W to the graph.
         * The return is for you convience and will not be graded.
         */
```

```
42          bool addEdge(const int start, const int end, const W &weight);
43
44          /*
45           * Remove edge from graph.
46           * The return is for you convience and will not be graded.
47           */
48          bool removeEdge(const int start, const int end);
49
50          void depthFirstTraversal(void (*visit)(const int node));
51          void breadthFirstTraversal(void (*visit)(const int node));
52
53          /*
54           * Return adjacent weight from start to end (or -1 if they are
55           * not adjacent.
56           */
57          W adjacent(const int start, const int end);
58
59          /* Run Dijkstra's Shortest Path to find the shortest path from start
60           * to end and returning that smallest weight.
61           * return -1 if a path does not exist!
62           */
63          W dijkstraShortestPath(const int start, const int end);
64
65          /* Print out the Graph */
66          void print() const;
67
68  };
69
70  #include "grapham.cpp"
71
72  #endif
```

Write some test cases:
Create some test cases, using Unity, that you believe would cover all aspects of your code.

Memory Management:
Now that are using new, we must ensure that there is a corresponding delete to free the memory. Ensure there are no memory leaks in your code! Please run Valgrind on your tests to ensure no memory leaks!

How to turn in:
Turn in via GitHub. Ensure the file(s) are in your directory and then:

- $ git add <files>
- $ git commit
- $ git push

Due Date: November 11, 2020 2359

Teamwork: No teamwork, your work must be your own.