

lab 22 Graphs via Adjacency Matrix

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

Implement the following class:

```
1 #ifndef GRAPHAM_H
2 #define GRAPHAM_H
3
4 /* This class represents a weighted directed graph via an adjacency matrix.
5  * Vertices are given an index, starting from 0 and ascending
6  * Class W : W represent the weight that can be associated with an edge.
7  * We will not weight the vertices.
8  * W is the data type for the weight. Normally an int.
9  */
10
11 template<class W>
12 class GraphAM {
13     private:
14         /* Recommended, but not necessary. */
15         void depthFirstTraversal(void (*visit)(const int node),
16                                 int *visited, const int cVertex);
17         /* You fill out private member variables. */
18     public:
19         /* Initialize an empty graph. */
20         GraphAM();
21
22         /* Initialize the Graph with a fixed number of vertices. */
23         GraphAM(const int vertices);
24
25         /* Destructor shall free up memory */
26         ~GraphAM();
27
28         /* Removes a vertex.
29          * return whether successful or not
30          * Note: You must shift all vertices accordingly.
31          */
32         bool removeVertex(int idx);
33
34         /* Adds amt vertices to the graph. Returns the starting point
35          * of the vertice count.
36          */
37         int addVertices(int amt);
38
39         /* Adds an edge with weight W to the graph.
40          * The return is for you convience and will not be graded.
41          */
```

```

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 cxxtestgen, 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: April 15, 2019 2359

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