Fernando Salazar

Final Projects (Graphs)

1. (20%) Create a design **before** you start coding that describes or shows how a graph structure could be used to store some kinds of data and attempt to solve some kind of problem (yes, this can be a game that needs a graph to represent a map!),

2. (20%) Create some tests (at least **two** for each piece of functionality) **before** you start coding...

 (40%) Implement a graph class with at least (this category effectively combines implementation and 3.specification, partly to emphasize getting the algorithms working!):

1. (5%) a function to add a new vertex to the graph (perhaps add\_vertex(vertex\_name)),
2. (5%) a function to add a new edge between two vertices of the graph (perhaps add\_edge(source, destination) or source.add\_edge(destination)),
3. (15%) a function for a shortest path algorithm (perhaps shortest\_path(source, destination)),
4. (15%) a function for a minimum spanning tree algorithm (example min\_span\_tree()).

4. (10%) Analyze the complexity of all of your graph behaviors (effectively a part of our documentation for grading purposes),

5. (10%) Once you have implemented and tested your code, add to the README file what line(s) of code or inputs and outputs show your work meeting each of the above requirements (or better, include a small screen snip of where it meets the requirement!).

1. Design Graph class

Going to need edges and vertices

It will need to have two algorithms, one using Dijkstra to find the shortest path and one for the minimum spanning tree.

Vertices

Add vertex function, name value

Edges

Add edge function, value for both beginning and end of edge, weight value for path.

Shortest Path

Function will need to have a starting node and an ending node

Some loop functions will go through all the edge weights to find the lowest path.

Minimum span tree

Function will need to have a starting node

Using prims algorithm

Check function

Check for vertex and edges

1. Testing for potential Functions

(two tests per function)

// add vertex function tests

// test one, add one vertex, and check it In.

A computer screen shot of text

Description automatically generated

// test two, add more than one vertex, and check they are all in.

A computer code on a black background

Description automatically generated

// add edges functions tests

// test one, add one edge, and check that it has been added.

A computer screen shot of a code

Description automatically generated

// test two, add more than one edge and check that they have been added.

A computer screen shot of a program code

Description automatically generated

//test for shortest path

//Test for a shorter path, maybe direct or two edges only.

A screen shot of a computer code

Description automatically generated

// test with longer paths and multiple paths.

A screenshot of a computer program

Description automatically generated

// test for minimum span tree

// a short or simple tree

A computer screen shot of a program

Description automatically generated

// a longer graph with more edges and

A computer screen shot of a program code

Description automatically generated

//Results

A computer screen with white text

Description automatically generated

1. Code for graph class.

// vertex function

A screen shot of a computer code

Description automatically generated

//edge function

A computer screen shot of a program code

Description automatically generated

//shortest path function

A computer screen with text on it

Description automatically generated

A screen shot of a computer program

Description automatically generated

// spanning tree function

A screen shot of a computer program

Description automatically generated

A computer screen with text on it

Description automatically generated

1. Code analysis (complexity)

// add edges

It should be O(1); it does not need to sort through the array just added; only check its size. Time never increases.

// add vertex

It should be O(n). It only needs to pass through once, but it will increase as it passes through the larger array.

//shortest path

I think it's O (n2). The process will repeat itself each time for each vertex, so the amount per vertex will increase. It has two parts that require going through the whole array to find a match.

// min span tree

O(n) for the main loop that needs to go through everything. It has a couple of parts. Setting the initial values will also cycle through but only at the very beginning.