Single Source Shortest Path - Dijsktra's Algorithm

1. Overview of Classes

• What class(es) did you design? - Describe each class in your design; explain the purpose of the class and of their member variables and member functions.

Class:

UndirectedGraph

Description:

Represents a Graph that stores information related to it's vertexes and edges as well as contains functionality to edit the graph and find the shortest weight between two vertexes.

Member variables:

- Vertex list
- Total of number of edges
- Total of number of vertexes

Member functions (operations) - description, parameters and return values

- insertVertex: Inserts vertexes into the graph include the info of the passed in name, fails if the vertex is already in the graph
- insertEdge: inserts an edge between the passed in vertexes and assigns it a weight, fails if the vertexes are the same or not in the graph or if the weight is not greater than zero
- searchVertex: searches for a passed in vertex in the graph, fails if the vertex is not found
- degree: returns the number of adjacent vertexes to a passed in vertex, fails if a vertex is not in the graph
- vertexCount: returns the total number of vertexes in the graph
- edgeCount: returns the total number of edges in the graph
- weight: gets the weight value between two edge values, fails if the passed in vertexes are not adjacent, uses a helper function to find the weight
- smallestWeight: finds the shortest weight between two vertexes, fails if the vertexes are the same, they are not connected by a path or if one of the vertexes is not in the graph, uses a helper function to calculate the shortest path
- printPath: prints the path of the shortest weight, fails if the vertexes are the same, they are not connected by a path or if one of the vertexes is not in the graph
- clear: removes all the edges and nodes from the graph
- relax: helper function to find the shortest path by comparing weights between two passed in vertexes and setting parent and key values of a Vertex as necessary
- removeZeros: removes trailing zeros from a string representing a double

Class:

PriorityQueue

Description:

Represents a binary min heap that can be used as a priority queue that contains nodes storing information about the vertexes from the graph.

Member variables:

- A set containing pointers to each Vertex
- Size of the heap

Member functions (operations) - description, parameters and return values

- extractMin: extracts Vertex with the minimum key from the queue
- modifyKey: sets the key of a passed in Vertex and resorts the heap based off the passed in value
- parent: calculates the index for the parent of a node based off the passed in index
- left: calculates the index of the left child of a node based off the passed in index
- right: calculates the index of the right child of a node based off the passed in index
- heapify: inserts a node at a specified index properly into the heap
- getHeapSize: returns the heap size

Class:

Vertex

Description:

Node to represent Vertex.

Member variables:

- Vertex Name
- Key
- Index of parent
- Index in the heap
- Vector of edges for adjacencies

Member functions (operations) - description, parameters and return values

- addEdge: adds an Edge to the adjacency list
- Functions to get and set the above variables from the Node class

Class:

Edge

Description:

Class to contain information about an edge.

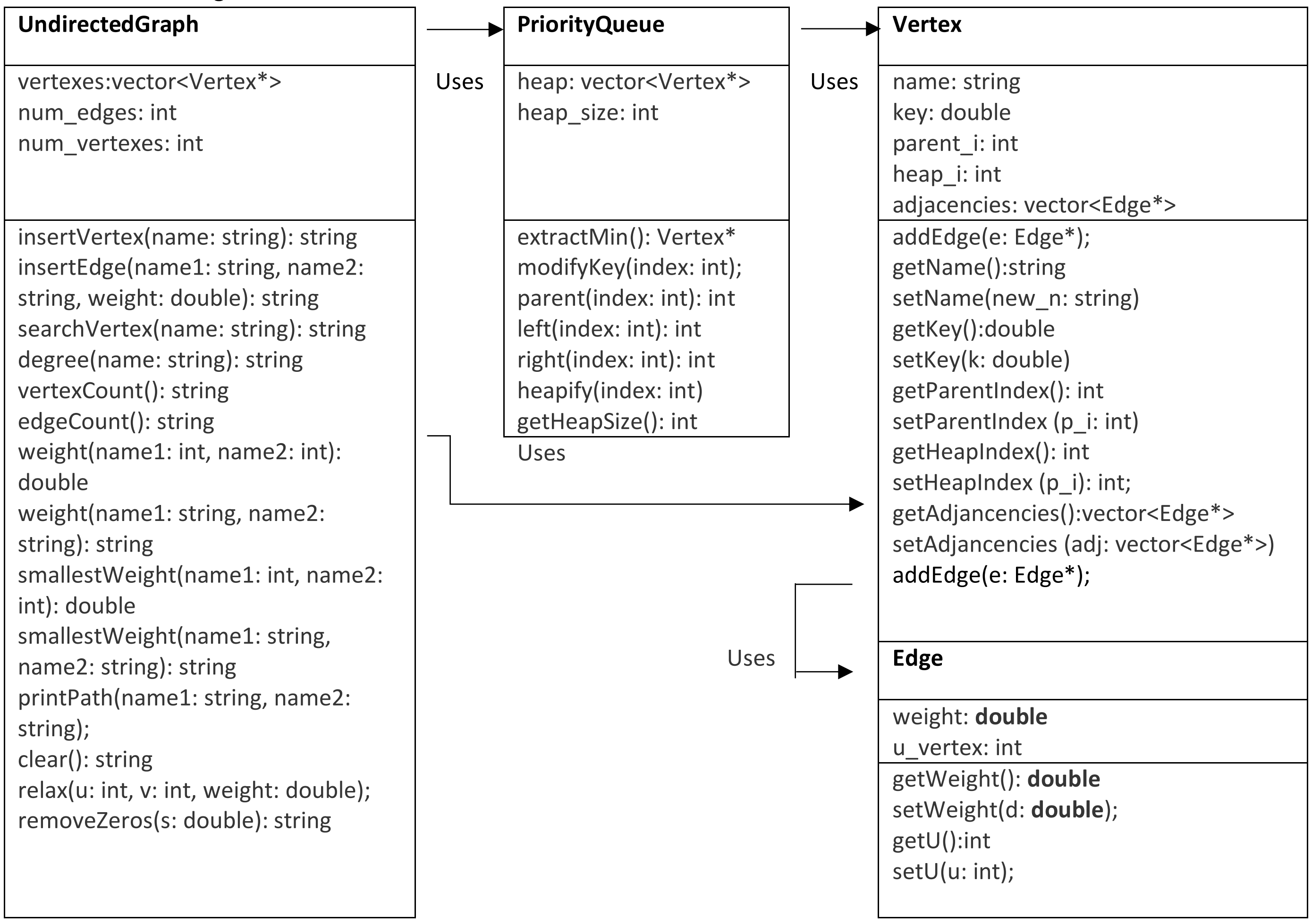
Member variables:

- weight
- index

Member functions (operations) - description, parameters and return values

- Functions to get and set the above variables from the Edge class

UML Class Diagram



2. Constructors/Destructor/Operator overloading

• For each class, what are your design decisions regarding constructors?
• For each class, what are your design decisions regarding destructors?
• Provide your rational for any operators that you needed or decided to override.

Graph:

The constructor will set the number of vertexes and edges to zero as well as create an empty vector of vertexes. The destructor calls the clear function. I did not find I needed to overwrite any operators for this class

PriorityQueue:

The constructor will create an empty heap. There will be another constructor that creates a binary min heap passed off a passed in set of vertexes. The destructor will set the pointer to each Vertex object to be null pointer. I did not find I needed to overwrite any operators for this class.

Vertex:

The constructor will set the name to an empty string, the indexes to -1 and the list of the adjacencies to and empty vector. There is also another constructor that sets the name to the passed in string, the indexes to -1 and the list of the adjacencies to and empty vector. The destructor will delete all the Edges and set the pointers to null pointer. I did not find I needed to overwrite any operators for this class.

Edge:

The constructor will set the weight and other the Vertex's index to default values. There is also another constructor that sets the weight and other the Vertex's index to passed in values. I did not find the need for a destructor. I did not find I needed to overwrite any operators for this class

3. Test Cases

• Describe your testing strategy for this class. What are some of the test cases you need to consider?

I tested each case where each test fails and then a scenario where each case should pass. I also checked whether other commands work after a fail. I tested adding vertexes and edges as well as clearing the whole graph. I tested running a clear when the graph was empty. I tested edge count when the graph is empty, after you add an edge and after you remove and edge. Called shortest_d on a connected and not connected graph and checked for the right answer. I also tested changing an edge weight once it had been added before.

4. Performance

I implemented the priority queue class as a binary heap as it allows for better overall time efficiency. In my graph I chose to contain my graph in an adjacency list as the graph is dynamically changing in size as you can insert new vertexes as well as edges and an adjacency list is a more efficient way to add vertexes.

Inserting a new node has O(1) time complexity as creating a Vertex and inserting the Vertex in the vector of Vertexes are both O(1) time. Inserting an edge into the graph is O(|V|) time complexity as it loops through every edge attached the Vertex to check if it was already in there before inserting the edge. The search function and the clear function has a time complexity of O(|V|) as it searches through every Vertex in the Vertex list. The degree and weight also have O(|V|) time complexity as they both call the search function which overpowers the rest of the command. The print path command has the same time complexity as the shortest weight function as it calls the command and then loops through the vertexes causing a time complexity of O(|E|log|V|).

Breaking up the Dijkstra function

- O(|V|) to set the default keys, parent indexes and heap indexes
- O(|V|) to create the priority queue as heapify is called for half the nodes the combination of both commands results in O(|V|) running time
- O(|V|log|V|) extractMin is called for each vertex and extractMin is O(log|V|) since it calls heapify at the root of the heap
- O(|E|log|V|) as modifyKey is called for every edge and modifyKey is O(log|V|) since it causes the traversal of the height of the heap

Overall Dijkstra time complexity is O(|V|log|V| +|E|log|V|) . In the average case ther are more edges then vertexes so the time complexity is O(|E|log|V|).