Joshua Barrientos
Cs-2400-02 9:30 - 10:35 MWF
Project 3
Due May 15, 2020

Project Specification:
getCheapestPath(T begin, T end, Stack<T> path) is used to find a path with a vertex with a label begin, and a vertex with a label end in a weighted Graph. It will return a double.

Method Summary(For GraphInterface)
All MethodsInstance MethodsAbstract Methods

Modifier and Type	Method	Description
boolean	**addEdge(T begin, T end)**	Adds an unweighted edge between two given distinct vertices that are currently in this graph.
boolean	**addEdge(T begin, T end, double edgeWeight)**	Adds a weighted edge between two given distinct vertices that are currently in this graph.
boolean	**addVertex(T vertexLabel)**	Adds a given vertex to this graph.
void	**clear()**	Removes all vertices and edges from this graph resulting in an empty graph.
double	**getCheapestPath(T begin, T end, Stack<T> path)**	Gets the shortest path of a weighted graph.
int	**getNumberOfEdges()**	Gets the number of edges in this graph.
int	**getNumberOfVertices()**	Gets the number of vertices in this graph.
boolean	**hasEdge(T begin, T end)**	Sees whether an edge exists between two given vertices.
boolean	**isEmpty()**	Sees whether this graph is empty.
boolean	**removeEdge(T begin, T end)**	Removes an existing edge from an distinct Vertices

Method Summary(For VertexInterface)

All MethodsInstance MethodsAbstract Methods

Modifier and Type	Method	Description
boolean	**connect(VertexInterface<T> endVertex)**	Connects this vertex and a given vertex with an unweighted edge.
boolean	**connect(VertexInterface<T> endVertex, double edgeWeight)**	Connects this vertex and a given vertex with a weighted edge.
boolean	**disconnect(VertexInterface<T> endVertex)**	Gets a vertex and edge and if it is in the list, it will remove it
double	**getCost()**	Gets the recorded cost of the path to this vertex.
T	**getLabel()**	Gets this vertex's label.
Iterator<VertexInterface<T>>	**getNeighborIterator()**	Creates an iterator of this vertex's neighbors by following all edges that begin at this vertex.
VertexInterface<T>	**getPredecessor()**	Gets the recorded predecessor of this vertex.
VertexInterface<T>	**getUnvisitedNeighbor()**	Gets an unvisited neighbor, if any, of this vertex.

Iterator<Double>	**getWeightIterator()**	Creates an iterator of the weights of the edges to this vertex's neighbors.
boolean	**hasNeighbor()**	Sees whether this vertex has at least one neighbor.
boolean	**hasPredecessor()**	Sees whether a predecessor was recorded for this vertex.
boolean	**isVisited()**	Sees whether the vertex is marked as visited.
void	**setCost(double newCost)**	Records the cost of a path to this vertex.
void	**setPredecessor(VertexInterface<T> predecessor)**	Records the previous vertex on a path to this vertex.
void	**unvisit()**	Removes this vertex's visited mark.
void	**visit()**	Marks this vertex as visited.

Project methodology:

I had tested out all of the different actions to take using my Main class Airport.java. The methods that needed to be tested out according to the project description in the DirectedGraph which were addEdge(), removeEdge(), and getCheapestPath() which was to be implemented using Dikstra’s algorithm. I first tested out instances of adding new Edges(), adding duplicate Edges(which printed out a warning message, removing an edge and removing an edge that wasn't there. I then traced a path and determined from that path and saw from the distance what was the shortest path. I cross referenced the two to determine whether I implemented the shortestPath of a weighted graph correctly. For The method to finding an airport name from an acronym or code, i had just placed two adjacent arrays together and search for a the acronym in its array of the airport to then get the name from the other array which would be at the same index.

Lessons Learned:

I learned how to correctly implement the getCheapestPath() from Dikstra’s algorithm for a Graph.