**Dijkstra Assignment Report**

Ali Hussein

**Introduction**

This assignment entailed implementing Dijkstra’s Shortest Path Algorithm into a program. The program should first of all read in a graph from a text file which holds either a stream of weighted edges or an edge-weighted array. I opted to go with the edge-weighted array. Once this file has been read, Dijkstra’s algorithm must be used on the read graph.

**Description**

The first step I went with when making this program was beginning on the reading a graph part. The instructions ask to test the implementation with a large graph. I didn’t want to go making a large graph line by line, so I decided to create a java class that writes a file. As shown in figure 1, I firstly make the target where I want the file created using the filename parameter that is taken in when calling this class. I also declare the random object to use later in this class when making the graph which I will explain in figure 2. I declare all my needed variables and then ask the user to input the source node and node amount for their graph.

Text

Description automatically generated

Figure - Taking in source and vertex amount input

This last part of the class will loop over how many nodes the user wants and create random values for the nodes from, target and weight. In my from node, I have a simple do while loop and if statement to make sure the first node in the graph is the source node although this is not essential.

Text

Description automatically generated

Figure - Creating random nodes in graph

After creating the WriteFile class, I moved onto actually starting the algorithm. I firstly created a Vertex class to hold a list of the vertices as shown in figure 3. This is a very simple static class which holds the vertex and its weight and also has get methods for these variables. This class is used in the Dijkstra class I will speak about in the next paragraph.

Text

Description automatically generated

Figure - Vertex class

The next class was the Dijkstra class which implemented the algorithm. As shown in figure 4, I start with creating a private static int variable named ‘NO\_PARENT’ and initialise it to -1. This is used when a node does not have a path. For example, the source will not have a path, so I initialise its parent path to NO\_PARENT or -1. After creating this variable, I begin my Dijkstra method which takes in the amount of vertices, an ArrayList using the Vertex class as its type and the source node. Then I create two arrays, distance[] will hold the shortest distance between the source and the target and parents[] will hold the path tree that the source took to get to the target. Now I create a for loop until the amount of vertices and make every index in distance a MAX\_VALUE. Then as I said before I make the source distance 0 and also its path that it takes -1.

Text

Description automatically generated

Figure - Dijkstra class declaring and initialising main variables

As shown in figure 5, I use a PriorityQueue with the Vertex class as the type. This will hold all the vertices I add into it in a natural order while comparing the edge weights. After creating it, I add the source node as the first element in the PriorityQueue. Now I can start my while loop to keep looping until the PriorityQueue is less than 0. Inside this while loop, I create a new vertex list called current and use PriorityQueue.poll() to take out the head of the PriorityQueue and place it in current so for the first iteration of the loop, source will go into current. I create an int nearest vertex to use when creating the path the node takes. Then I start my for loop to iterate over the Vertex ArrayList and inside I have an if statement to check the shortest distance and when it finds the shortest distance, it will add it to the distance[] array. The array parents[] is also used to get the path the source took to get to the target.

Text

Description automatically generated

Figure - Dijkstra class getting the shortest distance and path

The last part of this method was actually printing out the vertex, distance, and path for each node. As shown in figure 6, I use a for loop to iterate until it hits the length of distances[]. Then I use an if statement to make sure it doesn’t output the source to the source because that would be useless information on the screen. This will print the current vertexIndex then I use an if statement to print the shortest distance to get to the target. I use the if statement because some nodes are unreachable from the source so the program will spit out 2137483647 instead. If this happens, I don’t want this big number as it will be confusing to the user, so I just write null instead. Lastly, I print out the path it took to get to the target using the printPath() method in figure 7.

Text

Description automatically generated

Figure - Printing out stats

This method takes in the current vertex and an array then checks if the current vertex has no parents or equals -1. If true then returns nothing otherwise it will just call itself to print out its path.

Text

Description automatically generated

Figure - Printing out path

The final part of my program was reading the graph and implanting Dijkstra’s Algorithm. I firstly ask the user if they want to create their own random graph or use the set graph, graph2.txt. Then after bringing in the File object and initialising it with the users graph file, I create an ArrayList which will hold the vertices from the text file.

Text

Description automatically generated

Figure - Reading a file (1)

As shown in figure 9, it will firstly read in what the source node is, then the type of graph. As I have opted to use an edge weighted array graph, I made sure the file had ‘A’ as the type. Then I scanned the vertex amount and create a for loop to loop until the vertex amount is hit which will add to the arraylist indexes new ArrayLists inside each index. Then another while loop is used until there is nothing left to scan in the text file. Inside this while loop it will scan the from, target and weight then add it to the arraylist vertex. Once the loop is complete I call my Dijkstra class and use vertexAmount, the array list and the source as the parameters.

Text

Description automatically generated

Figure - Reading in graph

When running this code and making a graph of 50 vertices with a source vertex of 0 I got the output shown in figure 10. There are a lot of null distances as the source 0 cannot reach these nodes but some have a path from 0.

Text

Description automatically generated

Figure - Graph with 50 vertices