STATISTICS, ALGORITHMS & AI

CONTINUOUS ASSESSMENT I

JORGE BAPTISTA, K00265964

GAMES DESIGN AND DEVELOPMENT – BSC (HONS)

PROFESSOR: EUGENE KENNY

THURLES, NOVEMBER 2021

Contents

// Outline

In this assignment we were asked to implement the Dijkstra's algorithm to find the shortest paths between nodes in a graph. The process for me was a bit slow and even frustrating at first because I have decided to go with Java language and IntelliJ which I have never used before. IntelliJ was very straightforward and easy to set up and use. Java was a bit more of a struggle at first because even though it is very similar to C#, some syntax might change, and the built-in functions and libraries are different.

Something I have been learning to get used to is handwriting/drawing on paper and getting some pseudo code up. This helps a lot into getting a base for the algorithm ahead when turning this pseudo code into a lower-level version. I also had to search a lot on the internet to get help with java language itself with problems such as reading files, creating, and writing in files, parsing, etc.

Although the lecture pdfs have helped me a lot reminding me how the Dijkstra's Algorithm work I had to search many examples of the algorithm written in programming to get a general idea of what I must do and create a more complete pseudo code on paper for me to follow and then adapt to my own code. I have listed all (or most) of the websites that I have used for research for successfully writing my program and algorithm.

// Detailed Design

To start I followed the program we have done in class for the Graph and Edge class and slightly adapted into my own program (deleting the unused parts). Then to write the actual Dijkstra's algorithm I created a class for it and implemented the code in there while using main to call the function that solves the graph from it.

To create this algorithm, I had to search a lot for the best way to do it and wrote some pseudo code on paper which helped me immensely to get it all right.

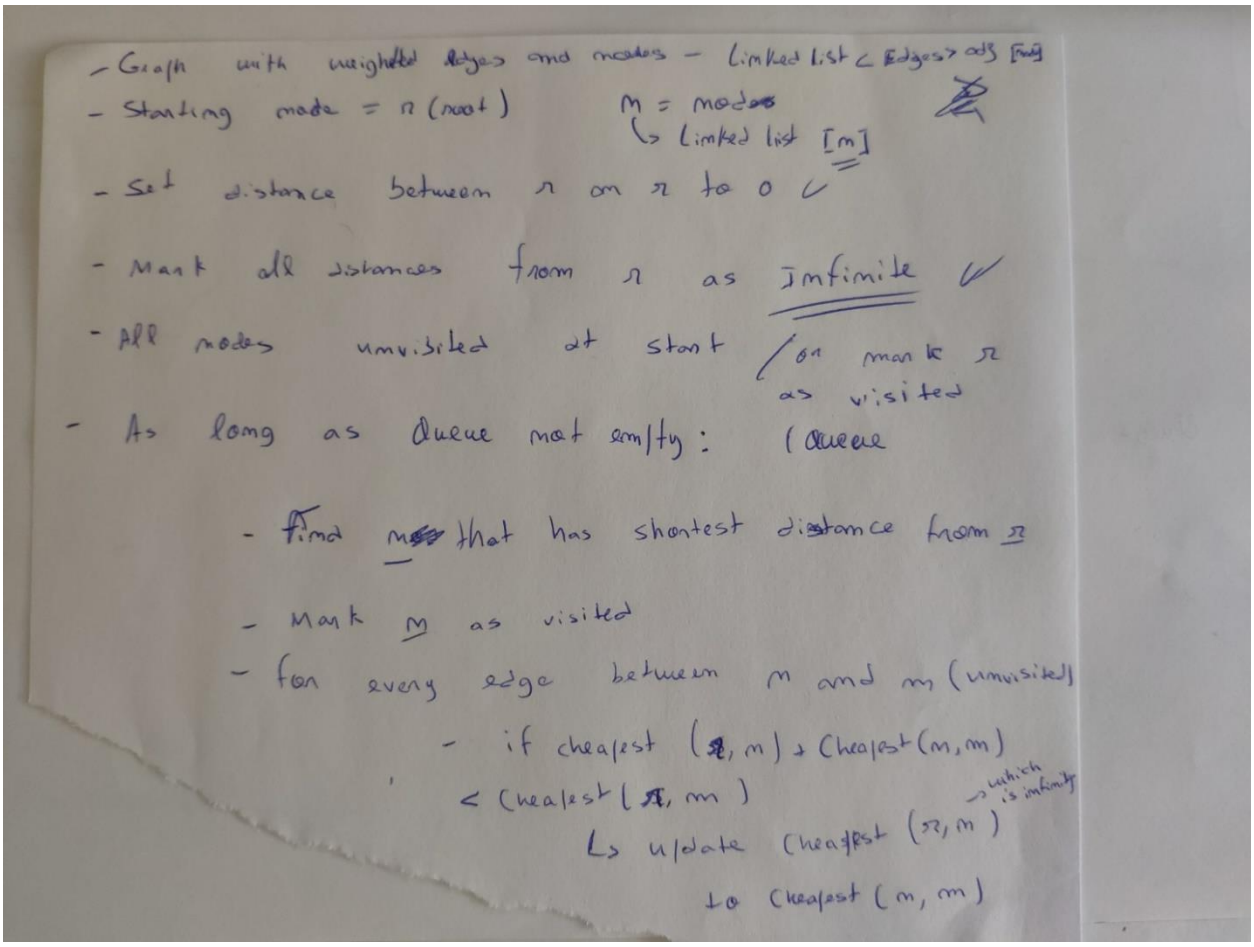

Figure a. High level pseudo code on paper.

Algorithm

So, following this pseudo code I then used the algorithm we used in class for DFS and adapted it for Dijkstra's as I did not want to copy other programs and wanted to also learn for myself while keeping the way of coding consistent to what we learned in class.

```
while (!queue.isEmpty()) // while queue is not empty
{
    int currentNode = queue.removeFirst(); // remove first node from queue and assign

    if (!visited.contains(currentNode)) // if this node was not visited
    {
        visited.add(currentNode); // add to visited

        for (Edge edge : g.edges(currentNode)) // for each edge this node has to
        {
            if (!visited.contains(edge.getDestination())) // if destination node was not visited
            {
                if (d[currentNode] + edge.weight < d[edge.destination])
                {
                    d[edge.destination] = d[currentNode] + edge.weight;
                    p[edge.destination] = currentNode;
                }

                queue.add(edge.getDestination());
            }
```

*Figure b. A preview of the algorithm. With **D** being distance and **P** being parent.*

It takes a root node as required in the assignment and then gets all the total weight that takes from this root to each other nodes. It also stores a parent array to print the path from root node to N node.

To simplify I wrote the printing part at the end of the program so that in main() only the graph and root node needs to be passed and then the function algorithm will do everything.

```
55      public void printDijkstra(int[] parent, int[] distance, int root, int nodes){
56          System.out.println("Dijkstra Algorithm: (With all paths)");
57          for (int i = 0; i < nodes ; i++) {
58              System.out.print(" " + root + " -> " + i + " = Distance : " + distance[i] + "  Path : ");
59              printPath(parent, i);
60              System.out.println();
61          }
62      }
63
64  @   public void printPath(int[] parent, int destination){
65          //if node is source then stop recursion
66          if(parent[destination] == -1) {
67              System.out.print("0 ");
68              return;
69          }
70  ↺       printPath(parent, parent[destination]);
71          System.out.print(destination + " ");
72      }
```

Figure c. Printing the outcome to the console.

Reading Input

To get the input from a file, with my current (little) knowledge of Java I used Scanner to read the file, get the first line to check if it is an S for a stream of arcs (if not the program stops there using the return keyword), the second for the total number of nodes that the created graph will have and then store the rest of the lines in a string for edges which are then converted to int when adding each edge to the graph. The function then creates a graph with the specified number of nodes and adds edge by edge to it. At the end it returns this newly created graph.

```
10          int nodes;
11          String[] edges;
12
13          try
14          {
15              File graphFile = new File(fileName);
16              Scanner fileReader = new Scanner(graphFile);
17
18              if (fileReader.nextLine().compareTo("S") == 0)
19              {
20                  System.out.println("Reading a stream of weighted arcs...");
21              }
22              else
23              {
24                  System.out.println("This program only accepts a stream of weighted arcs for input");
25                  return null;
26              }
27
28              nodes = Integer.parseInt(fileReader.nextLine());
29
30              List<String> lines = new ArrayList<>();
31
32              while (fileReader.hasNextLine())
33              {
34                  lines.add(fileReader.nextLine());
35              }
36
37              edges = lines.toArray(new String[lines.size()]);
```

Figure e. Reading all lines and storing them.

```
46          Graph g = new Graph( n: nodes + 1);
47
48          // add edges
49          for (int i = 0; i < edges.length - 1; i++)
50          {
51              String[] splitted = edges[i].split( regex: "\\s+"); // split each line per white space
52              int[] numbers = new int[splitted.length]; // create an int array to store the converted values
53              for (int j = 0; j < splitted.length; j++)
54              {
55                  numbers[j] = Integer.parseInt(splitted[j]); // convert value to int and store in int array
56              }
57              g.addEdge(numbers[0], numbers[1], numbers[2]); // create the edges
58          }
```

Figure d. Creating a graph and adding edges to it based on the data stored from the lines of the file.

Main function

Then on the main function this readFile() is called and the name of the file passed in (e.g. “SparseGraph.txt”) which then returns a graph object from it. Then the program asks for user input to enter the root node which all paths and their weights will be printed from and then calls the Dijkstra algorithm to solve it using this graph.

To improve user interaction and minimize errors values were checked like for example only allowing the used to select a root node from 0 to the total number of nodes on that graph.

```
75      public static void main(String [] args)
76      {
77          ReadFile readFile = new ReadFile();
78
79          Graph g = readFile.generateGraph( fileName: "SparseGraph.txt");
80
81          if (g == null) return; // if there is no graph returned from ReadFile, end program
82
83          int nodes = g.n - 1; // total number of nodes
84
85          // get user input for the root node
86          Scanner reader = new Scanner(System.in);
87          System.out.println("Number of nodes in this graph: " + nodes);
88          System.out.println("Enter the root node: ");
89
90          int r = -1; // initialize with a wrong value
91
92          // while the input is not between 0 and the total number of nodes
93          while (r < 0 || r > nodes)
94          {
95              r = reader.nextInt();
96
97              if (r < 0 || r > nodes)
98              {
99                  System.out.println("Please enter a number between 0 and " + nodes + "."); // error check
100             }
101         }
```

Figure g. Passing a file to read and getting a generating a graph from it and then asking user input for the root node.

```
103             // run Dijkstra algorithm
104             Dijkstra dijkstra = new Dijkstra();
105             dijkstra.dijkstra(g, r);
106     }
```

Figure f. Running the algorithm using the created graph and root node.

Many comments were added along the code to make it more readable for others.

// Unit Test Plan

To test the algorithm, I used the data from different websites and compare the output of those websites with the output of my program. The expected result was for both outputs to be equal or similar and it was. At first I was afraid I would have errors or problems and the output would not be the same. But using the diverse information I had to create a good algorithm (lecture pdf and online explanations and examples) allowed me to succeed in this task.

To then test the algorithm with larger files as asked on the assignment I created a class to generate a graph file based on different parameters such as minimum nodes, maximum nodes, probability (of edges to appear between nodes where more probability makes it denser) and name of the file.

```
13      public RandomGraph(int min, int max, int sparsity, String fileName)
14      {
15          // properties of the file in the constructor
16          this.nMin = min;
17          this.nMax = max;
18          this.sparsity = sparsity;
19          this.fileName = fileName;
20      }
21
22      public void generateGraph() {
23          try {
24              File graphFile = new File(fileName);
25              if (graphFile.createNewFile()) {
26                  System.out.println("File created: " + graphFile.getName());
27              } else {
28                  System.out.println("File already exists.");
29              }
30          } catch (IOException e) {
31              System.out.println("An error occurred.");
32              e.printStackTrace();
33          }
```

Figure h. Getting the graph properties and creating a file for it.

```
        try {
            FileWriter graphFile = new FileWriter(fileName);

            graphFile.write( str: "S\n");

            Random random = new Random();
            int n = random.nextInt( bound: nMax - nMin + 1) + nMin;

            graphFile.write( str: n + "\n");

            for (int i = 0; i < n; i++) {
                for (int j = i + 1; j <= n; j++) {
                    if (random.nextInt( bound: 100) > sparsity) // sparse - less probability = more density
                    {
                        // for each node iterated through all nodes after it and has a probability to create an edge
                        graphFile.write( str: i + " " + j + " " + (random.nextInt( bound: 16) + 1) + "\n");
                        graphFile.flush();
                    }
                }
            }
            graphFile.close();

        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
```

*Figure i. Writing the first 2 lines as **S** for stream of arcs and then the total number of nodes in the graph. For each node have a probability to create an edge of x weight between all other nodes after it.*

In the main function of this class the generate graph file function is called and 2 files are used to test the algorithm, a large sparse graph, and a large dense graph.

```
    public static void main(String[] args)
    {
        RandomGraph sparseGraph = new RandomGraph( min: 1,  max: 1000,  sparsity: 90,  fileName: "SparseGraph.txt");
        sparseGraph.generateGraph();

        RandomGraph denseGraph = new RandomGraph( min: 1,  max: 1000,  sparsity: 20,  fileName: "DenseGraph.txt");
        denseGraph.generateGraph();
    }
}

```

Figure j. Generate 2 graphs for testing, a dense and sparse.

// Outcome

```
Reading a stream of weighted arcs...
Number of nodes in this graph: 831
Enter the root node:
345
Dijkstra Algorithm: (With all paths)
 345 -> 0 = Distance : 21  Path : 0 28 670 300 155 0
 345 -> 1 = Distance : 18  Path : 0 105 768 514 1
 345 -> 2 = Distance : 19  Path : 0 105 27 737 365 2
 345 -> 3 = Distance : 10  Path : 0 105 29 627 3
 345 -> 4 = Distance : 28  Path : 0 497 269 4
 345 -> 5 = Distance : 11  Path : 0 105 609 5
 345 -> 6 = Distance : 21  Path : 0 556 817 180 6
 345 -> 7 = Distance : 31  Path : 0 556 46 356 7
 345 -> 8 = Distance : 15  Path : 0 497 531 121 8
 345 -> 9 = Distance : 16  Path : 0 28 178 259 9
 345 -> 10 = Distance : 21  Path : 0 105 768 417 10
 345 -> 11 = Distance : 21  Path : 0 497 827 227 11
 345 -> 12 = Distance : 29  Path : 0 772 651 12
 345 -> 13 = Distance : 19  Path : 0 497 13
 345 -> 14 = Distance : 24  Path : 0 772 684 701 14
 345 -> 15 = Distance : 31  Path : 0 28 178 740 15
 345 -> 16 = Distance : 17  Path : 0 105 318 506 16
 345 -> 17 = Distance : 20  Path : 0 105 27 737 17
 345 -> 18 = Distance : 26  Path : 0 675 465 18
 345 -> 19 = Distance : 27  Path : 0 568 94 19
 345 -> 20 = Distance : 12  Path : 0 105 29 165 20
 345 -> 21 = Distance : 28  Path : 0 772 159 445 586 21
```

Figure k. Example of the first lines of the output using one of the randomly generated graphs.

// Conclusion

This project not only taught me a lot about algorithms by getting hands on with the problems and trying to solve them but also gave me a great opportunity to introduce myself to Java language and IntelliJ software.

This was a challenge, but I am very happy with the outcome and was able to write the algorithm successfully (at least I think/hope so, after trying with different inputs) and deliver other requirements such as input and output rules of the program. And after this I will be much more ready for algorithms and Java language if I ever intend to learn more about it.

// References

For the Dijkstra's Algorithm:

- https://eugkenny.github.io/COMP7028/lectures/Algs%20-%20Lecture%2004%20-%20Graphs%20-%20Shortest%20Paths.pdf
- https://stackabuse.com/graphs-in-java-dijkstras-algorithm/
- https://algorithms.tutorialhorizon.com/dijkstras-shortest-path-algorithm-spt-adjacency-list-and-priority-queue-java-implementation/
- https://algorithms.tutorialhorizon.com/print-all-paths-in-dijkstras-shortest-path-algorithm/

For writing files in Java:

- https://www.programcreek.com/2011/03/java-write-to-a-file-code-example/
- https://www.baeldung.com/java-write-to-file

Reading files:

- https://www.w3schools.com/java/java_files_read.asp
- https://www.baeldung.com/java-read-line-at-number
- https://www.javatpoint.com/how-to-read-file-line-by-line-in-java
- https://www.daniweb.com/programming/software-development/threads/291207/compare-string-with-text-file-lines

Converting and splitting strings:

- https://stackoverflow.com/questions/18838781/converting-string-array-to-an-integer-array
- https://stackoverflow.com/questions/7899525/how-to-split-a-string-by-space/7899558
- https://www.educative.io/edpresso/how-to-convert-a-double-to-int-in-java

Random and probability in Java:

- https://stackoverflow.com/questions/8183840/probability-in-java

Getting User Input:

- https://stackoverflow.com/questions/5287538/how-to-get-the-user-input-in-java