CIS 405/505 ALGORITHM ANALYSIS AND DESIGN

Network Routing and Graph Paths
Project 2

Submitted by:

MANJULA DHONDIBA KHOT	USN: 73764162
ALEXANDER COLLINS	USN: 25562616

Network Routing and Graph Paths

TABLE OF CONTENTS

TABLE OF FIGURES

1.Overview

The project's goal was to correctly implement Dijkstra's algorithm for shortest paths, show each sub paths' cost, the total path cost, and the time required to execute it all. The implementation was all done completely in a single Java file, and was tested with a dataset provided publicly by Stanford University. Results were taken from a varying number of vertices, as well as from different source vertices. The number of vertices, outgoing vertices, source vertex, and execution time were all noted. Finally, we provide the time and space complexity of the algorithm.

2.Dijkstra's Algorithm

Dijkstra's algorithm is used to find shortest paths between nodes in a connected graph with weighted edges. It was named after Edsger Wybe Dijkstra, who wrote a note describing such a solution[1]. This is applicable to real world problems, where nodes represent facilities or locations, and edges represent the path that connect them. One real world application of Dijkstra's algorithm is GPS, where the algorithm is used to find the shortest path between the source and destination, with various different roads to take, alongside stops and slowdowns.

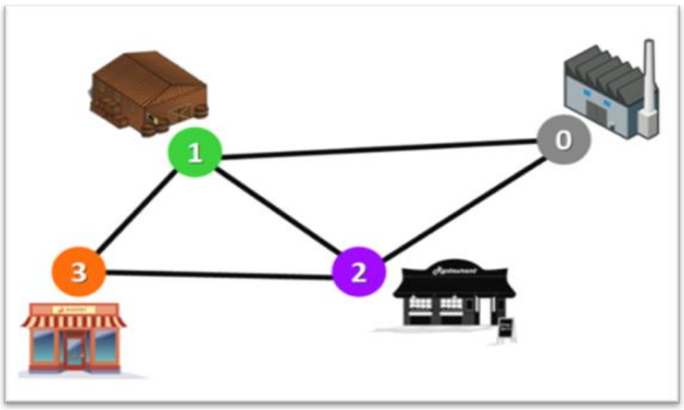


Figure 1: Example of Network represented by a Graph

[1] Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* 1, 269–271 (1959). https://0-doi-org.wizard.umd.umich.edu/10.1007/BF01386390

The way Dijkstra’s algorithm works is simple. It starts at the node chosen as the source and analyzes the complete graph to find the shortest distance between the source node and all other nodes in the graph. Once the shortest distance between the source and another node is determined, it adds that node to the path and the node is marked as visited. This process continues until all nodes are added to the path, giving the shortest path that connects the source node to all of the nodes.

3.Problem Statement

We were tasked with implementing Dijkstra’s algorithm. The goal was to not only get the same solution as Dijkstra’s algorithm would provide, but to also have the same process, checked through each sub path’s cost.

4.Implementation code of Dijkstra Algorithm

4.1. Code

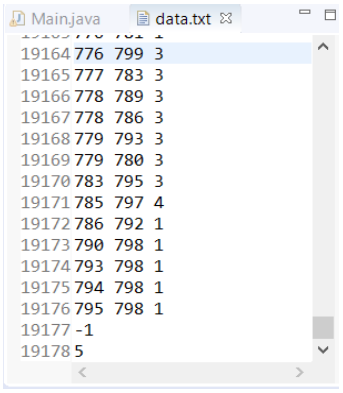
The execution of the code is kept within a try-catch to make sure there are no errors reading the input file. After reading in every line from the .txt file, the code checks for one of three possibilities with each line: the line is empty, in which case the code ignores; the line has coordinates for an edge (origin vertex, destination vertex, and weight), where the vertices and edge are added to the graph; and the line only has “-1”, indicating the end of the graph, at which the code identifies the source vertex for the graph (indicated on the line after), prints the graph, and finds the shortest path.

The function for finding the shortest path first takes the source vertex, sets its distance from the source vertex to zero (since it is the source itself), and adds it to a queue. Then, the program goes through each edge in the graph. If the edge has not been accounted for yet, the weight of the path thus far is calculated. If the path’s weight is less than the weight of any other path calculated thus far from the source vertex to the vertex at the end of the edge, then the current path becomes the shortest path from the source vertex, and is added to the queue.

4.2. Data

The dataset selected for the algorithm is from a standard set of graphs G-Set[2].

The figure below shows a few records from the dataset. The first column represents the first vertex, the second column represents the second vertex, and the third column represents the weight between the two vertices.



```
776 799 3
777 783 3
778 789 3
778 786 3
779 793 3
779 780 3
783 795 3
785 797 4
786 792 1
790 798 1
793 798 1
794 798 1
795 798 1
-1
5
```

Figure 2: Dataset

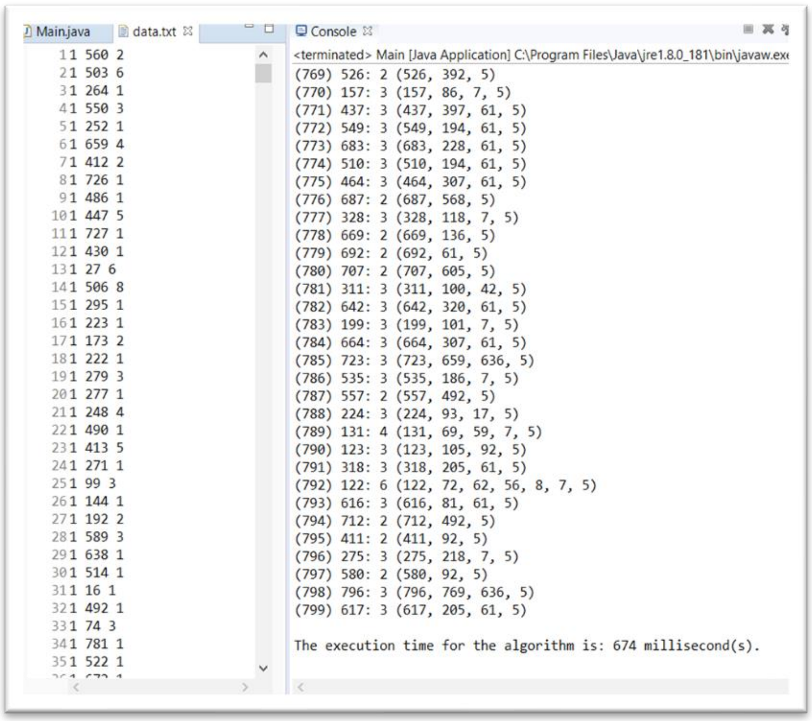
The '-1' represents the end of records for the dataset. The source vertex (shown as 5 in Figure 2) is given at the end of the dataset. The algorithm tests changing the weights in the third column to check the effect on the execution time.

5.Results

5.1. Solution

The figure below shows the results obtained by running the algorithm for the dataset with 798 vertices and 5 as the source vertex.

[2] Ye, Y. /~yyye/yyye/Gset/maxG11. https://web.stanford.edu/~yyye/yyye/Gset/maxG11



```
1 560 2
1 503 6
1 264 1
1 550 3
1 252 1
1 659 4
1 412 2
1 726 1
1 486 1
1 447 5
1 727 1
1 430 1
1 27 6
1 506 8
1 295 1
1 223 1
1 173 2
1 222 1
1 279 3
1 277 1
1 248 4
1 490 1
1 413 5
1 271 1
1 99 3
1 144 1
1 192 2
1 589 3
1 638 1
1 514 1
1 16 1
1 492 1
1 74 3
1 781 1
1 522 1
```

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe
(769) 526: 2 (526, 392, 5)
(770) 157: 3 (157, 86, 7, 5)
(771) 437: 3 (437, 397, 61, 5)
(772) 549: 3 (549, 194, 61, 5)
(773) 683: 3 (683, 228, 61, 5)
(774) 510: 3 (510, 194, 61, 5)
(775) 464: 3 (464, 307, 61, 5)
(776) 687: 2 (687, 568, 5)
(777) 328: 3 (328, 118, 7, 5)
(778) 669: 2 (669, 136, 5)
(779) 692: 2 (692, 61, 5)
(780) 707: 2 (707, 605, 5)
(781) 311: 3 (311, 100, 42, 5)
(782) 642: 3 (642, 320, 61, 5)
(783) 199: 3 (199, 101, 7, 5)
(784) 664: 3 (664, 307, 61, 5)
(785) 723: 3 (723, 659, 636, 5)
(786) 535: 3 (535, 186, 7, 5)
(787) 557: 2 (557, 492, 5)
(788) 224: 3 (224, 93, 17, 5)
(789) 131: 4 (131, 69, 59, 7, 5)
(790) 123: 3 (123, 105, 92, 5)
(791) 318: 3 (318, 205, 61, 5)
(792) 122: 6 (122, 72, 62, 56, 8, 7, 5)
(793) 616: 3 (616, 81, 61, 5)
(794) 712: 2 (712, 492, 5)
(795) 411: 2 (411, 92, 5)
(796) 275: 3 (275, 218, 7, 5)
(797) 580: 2 (580, 92, 5)
(798) 796: 3 (796, 769, 636, 5)
(799) 617: 3 (617, 205, 61, 5)

The execution time for the algorithm is: 674 millisecond(s).
```

Figure 3: Results

For each vertex, the algorithm displays the cost of the shortest path to the source vertex, and displays the path used by the vertex to reach the source vertex.

5.2. Execution Time for Different Instances of Data

The table below shows the execution time for the algorithm for different instances data. The execution time changes as the number of vertices, number of outgoing vertices from each vertex, source vertex, and weight of the edge change.

No. of vertices	No. of outgoing vertices	Source vertex	Execution time (sec)
798	variable	1	819
7	4	2	16
798	variable	3	696
50	variable	25	25
798	variable	700	611

Table 1: Dataset records

6.Time Complexity and Space Complexity

The algorithm takes values in the form of three rows, where the first row and second row represent two vertices, and the third row represents the weight of the edge between the two vertices.

6.1. Space Complexity

Space complexity is the storage space the algorithm needs in relation to its inputs.
The number of vertices in the graph is V. The average space complexity is O(V). The worst-case complexity when the graph is complete is O(V^2).

6.2. Time Complexity

Time complexity is the computer time used by the algorithm to run.
The time complexities are calculated by calculating the complexities of the function shortestPath().
If v = number of vertices and e = number of edges then,

- All the vertices of the graph can be traversed in O(V+E) time.
- Operations in heap like extracting min values and decrease-key values takes O(logV) time.
- Overall time complexity is O(E+V) * O(logV) which is O((E+V)*logV) = O(ElogV)
- Reduced time complexity: O(E+VlogV).

7.References

1. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* 1, 269–271 (1959). https://0-doi-org.wizard.umd.umich.edu/10.1007/BF01386390
2. Ye, Y. /~yyye/yyye/Gset/maxG11. https://web.stanford.edu/~yyye/yyye/Gset/maxG11
3. Estefania Cassingena Navone.[Figure 1] Retrieved from Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction (freecodecamp.org)