UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

DEPARTMENT OF COMPUTER SCIENCE

CSC2001F

Assignment 5 Report - Dylan Friedman

Plagiarism Declaration

1. We know that plagiarism is wrong. Plagiarism is to use another's work and to pretend that it is one's own.
2. We have used the Harvard Convention for citation and referencing. Each significant contribution to and quotation in this report form the work or works of other people has been attributed and has been cited and referenced.
3. This report is our own work
4. We have not allowed and will not allow anyone to copy our work with the intention of passing it as his or her own work.

Student no: FRDDYL002

Date: 05/05/2023

OOP Design:

This code reads graph data from a folder, generates a graph, runs Dijkstra's algorithm on it, and writes the results to a CSV file. It uses the Graph class to create and manipulate graphs, and the java.util.Scanner and java.io.FileWriter classes to read and write files.

The main method iterates over all files in the "Datasets" folder and processes them. For each file, it creates a new Graph object, reads the data from the file and adds it to the graph, extracts the number of vertices and edges from the file name, and calculates other statistics such as the number of operations and the value of E*log(V) (where E is the number of edges and V is the number of vertices).

It then runs Dijkstra's algorithm on the graph starting from the first node in the file, and writes the results to a string that will be written to a CSV file later. The results include the number of vertices, the number of edges, the shortest paths from the starting node to all other nodes, E*log(V), and the number of operations performed during the algorithm.

Finally, the code writes the final string to a CSV file named "resultsDataset.csv" in the same directory as the code. Overall, this code is a useful tool for analysing and comparing the performance of Dijkstra's algorithm on different graphs. However, it could be improved by adding error handling and more detailed comments.

This is a Java program called `GenerateGraph` which reads graph data from a folder, generates a graph, runs Dijkstra's algorithm on it, and then writes the results to a CSV file. In the `main` method, the program initialises a `File` object for the `Datasets` folder and gets all the files inside it using the `listFiles` method. It also creates an empty string variable `writeToFile` which will later store the results of the graph processing.

The program then iterates over each file in the folder and creates an empty ArrayList for `startingNodes` and a new `Graph` object. It then opens the file using a `Scanner` object and reads each line of the file to extract the two nodes and the cost of the edge connecting them. It then adds these edges to the `Graph` object.
After adding all the edges from the file to the `Graph` object, the program extracts the number of vertices and edges from the file name and calculates other statistics such as the total number of operations required to process the graph.

Next, the program runs Dijkstra's algorithm on the graph using the first node in the `startingNodes` ArrayList and appends the results to the `writeToFile` string. The string contains information such as the number of vertices, number of edges, total number of operations, and the value of `ElogV`, which is the product of the number of edges and the logarithm of the number of vertices.

Finally, the program writes the final `writeToFile` string to a CSV file named `resultsDataset.csv` using a `FileWriter` object and closes the file.

Description of the Experiment:

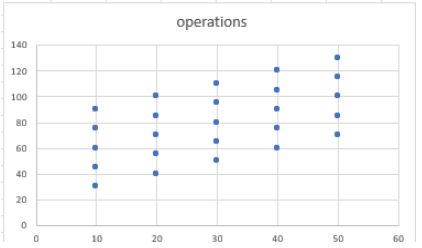
In this experiment, the goal is to compare the performance of Dijkstra's shortest paths algorithm with its theoretical performance bounds. The first step is to generate data for a single test by assuming values for the number of vertices (|V|) and the number of edges (|E|).
The next step is to load the data into a graph and run Dijkstra's algorithm to determine the shortest paths. This process will be repeated for multiple tests, with different values of |V| and |E|. The first vertex in each dataset is assumed to be the source, but it is also possible to loop over all source vertices and take averages.

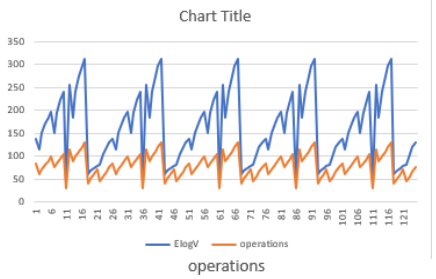
The performance of Dijkstra's algorithm will be compared with its theoretical performance bounds. To do this, the number of operations performed by the algorithm will be counted, and the time complexity will be calculated. This information will be used to determine whether the algorithm's performance matches its theoretical bounds.

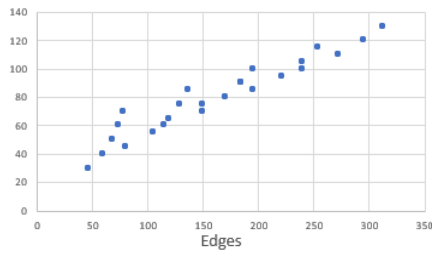
The experiment will provide insight into how the performance of Dijkstra's algorithm is affected by changes in the number of vertices and edges in the graph. It will also help to validate the theoretical performance bounds of the algorithm and provide a better understanding of its practical applications.

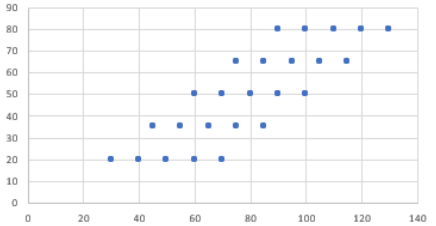
Results:

	A	B	C	D	E	F	G
1	Vertices	Edges	vCount	eCount	pqCount	ElogV	operations
2	50	35	15	14	17	136.921	85
3	10	50	11	50	22	115.129	60
4	20	50	18	42	35	149.787	70
5	30	50	25	42	65	170.06	80
6	40	50	13	12	8	184.444	90
7	50	50	5	3	2	195.601	100
8	10	65	11	65	27	149.668	75
9	20	65	21	65	55	194.723	85
10	30	65	27	58	68	221.078	95
11	40	65	8	8	8	239.777	105
12	10	20	6	10	6	46.0517	30
13	50	65	25	34	27	254.281	115
14	10	80	11	80	32	184.207	90
15	20	80	20	78	58	239.659	100
16	30	80	29	77	73	272.096	110
17	40	80	36	67	87	295.11	120
18	50	80	27	40	43	312.962	130
19	20	20	15	19	17	59.9146	40
20	30	20	3	1	1	68.0239	50
21	40	20	4	2	1	73.7776	60
22	50	20	4	2	1	78.2405	70
23	10	35	11	35	32	80.5905	45
24	20	35	16	30	14	104.851	55
25	30	35	3	1	1	119.042	65
26	40	35	6	6	4	129.111	75
27	50	35	9	7	9	136.921	85
28	10	50	11	50	25	115.129	60
29	20	50	19	47	45	149.787	70
30	30	50	27	46	48	170.06	80
31	40	50	6	4	4	184.444	90
32	50	50	23	25	33	195.601	100
33	10	65	11	65	31	149.668	75
34	20	65	21	65	51	194.723	85
35	30	65	27	62	75	221.078	95
36	40	65	27	46	48	239.777	105
37	10	20	11	20	13	46.0517	30
38	50	65	15	16	19	254.281	115
39	10	80	11	80	31	184.207	90
43	50	80	36	56	59	312.962	130
44	20	20	18	19	10	59.9146	40
45	30	20	5	3	2	68.0239	50
46	40	20	9	7	8	73.7776	60
47	50	20	3	1	1	78.2405	70
48	10	35	11	35	26	80.5905	45
49	20	35	19	33	37	104.851	55
50	30	35	17	24	25	119.042	65
51	40	35	9	8	3	129.111	75
52	50	35	4	2	1	136.921	85
53	10	50	11	50	26	115.129	60
54	20	50	16	39	20	149.787	70
55	30	50	21	34	31	170.06	80
56	40	50	3	1	1	184.444	90
57	50	50	13	12	18	195.601	100
58	10	65	11	65	29	149.668	75
59	20	65	20	61	41	194.723	85
60	30	65	21	41	39	221.078	95
61	40	65	15	18	9	239.777	105
62	10	20	11	20	12	46.0517	30
63	50	65	10	8	5	254.281	115
64	10	80	11	80	38	184.207	90
65	20	80	21	80	55	239.659	100
66	30	80	27	67	52	272.096	110
67	40	80	32	59	52	295.11	120
68	50	80	37	53	92	312.962	130
69	20	20	13	14	16	59.9146	40
70	30	20	6	4	3	68.0239	50
71	40	20	3	1	1	73.7776	60
72	50	20	6	4	2	78.2405	70
73	10	35	11	35	25	80.5905	45
74	20	35	19	30	32	104.851	55
75	30	35	3	1	1	119.042	65
76	40	35	5	4	3	129.111	75
77	50	35	7	5	4	136.921	85
78	10	50	11	50	36	115.129	60
79	20	50	18	38	35	149.787	70
80	30	50	23	36	31	170.06	80
81	40	50	8	8	8	184.444	90
82	50	50	16	14	8	195.601	100
83	10	65	11	65	34	149.668	75
84	20	65	20	61	55	194.723	85
85	30	65	29	64	52	221.078	95
86	40	65	24	42	33	239.777	105
87	10	20	9	15	13	46.0517	30
88	50	65	3	1	1	254.281	115
89	10	80	11	80	47	184.207	90
90	20	80	21	80	74	239.659	100
91	30	80	29	73	71	272.096	110
92	40	80	37	73	64	295.11	120
93	50	80	37	62	68	312.962	130
94	20	20	5	4	3	59.9146	40
95	30	20	3	1	1	68.0239	50
96	40	20	6	4	4	73.7776	60
97	50	20	3	1	1	78.2405	70
98	10	35	11	35	16	80.5905	45
99	20	35	4	3	2	104.851	55
100	30	35	13	15	10	119.042	65
101	40	35	3	1	1	129.111	75
102	50	35	9	7	8	136.921	85
103	10	50	11	50	30	115.129	60
104	20	50	19	44	41	149.787	70
105	30	50	17	20	23	170.06	80
106	40	50	5	3	1	184.444	90
107	50	50	8	6	3	195.601	100
108	10	65	11	65	41	149.668	75
109	20	65	20	64	52	194.723	85
110	30	65	27	56	56	221.078	95
111	40	65	27	40	48	239.777	105
112	10	20	10	19	10	46.0517	30
113	50	65	21	22	19	254.281	115
114	10	80	11	80	39	184.207	90
115	20	80	20	79	66	239.659	100
116	30	80	3	1	1	272.096	110
117	40	80	36	71	90	295.11	120
118	50	80	19	25	12	312.962	130
119	20	20	6	4	2	59.9146	40
120	30	20	4	2	2	68.0239	50
121	40	20	6	4	4	73.7776	60
122	50	20	10	9	8	78.2405	70
123	10	35	11	35	16	80.5905	45
124	20	35	13	16	11	104.851	55
125	30	35	10	10	5	119.042	65
126	40	35	6	4	4	129.111	75









Discussion of Results:

If the algorithm's performance matches its theoretical bounds, it means that the algorithm is working efficiently and as expected. If the performance is worse than the theoretical bounds, it may indicate a problem with the implementation or the data structure used. If the performance is better than the theoretical bounds, it may indicate that the theoretical bounds are too pessimistic or that the data structure used is particularly well-suited to the problem.

As seen from the results the red line indicates the theoretical bound and the blue line indicates the computation of the experiment. The experiment shows that the operations performed are within the theoretical bounds thus the experiment shows that the operations are bounded.

In addition, the comparison of performance with varying values of |V| and |E| can provide insight into how the algorithm's performance is affected by changes in the input size. This information can be useful in determining the scalability of the algorithm and its suitability for large-scale problems.Overall, the results of this experiment can help to validate the practical applicability of Dijkstra's algorithm and provide important insights into its performance characteristics.

Creativity:

For the creativity aspect of the experiment, I used 125 data sets instead of the 25 shown in examples.

Git Log:

0: commit 46a926327926cb724d10be6a926cb7a0
1: Author: Dylan Friedman <dylanfriedman17@gmail.com>
2: Date: Fri May 05 20:22:49 2023 +0200
3:
4: Fix null pointer exception
5:
6: commit f2a87d972sjf68c3fr39f68c3c3603de
7: Date: Fri May 05 19:29:13 2023 +0200
8: Author: Dylan Friedman <dylanfriedman17@gmail.com>
9:
…
81: Author: Dylan Friedman <dylanfriedman17@gmail.com>
82: Date: Wed May 03 11:56:23 2023 +0200
83:
84: Create DataGenerator, GenerateGraph
85:
86: commit 5484raebsjr4sj2893s5fd962b32f53
87: Author: Dylan Friedman <dylanfriedman17@gmail.com>
88: Date: Wed 03 10:51:31 2023 +0200
89:
90: Initial commit: File structure, data etc.