CSC2001F Report

Christopher Blignaut

BLGCHR003

Object Oriented Design

I used an object oriented design when coding this project to allow for the reusability of code. In doing so I created the class GraphGenerator.

This class included the static methods:

- generateConnections which created the edges (consisting of 2 letters as the nodes and an int used to be the weight for the edge) used to build the graphs and to be read in by the Graph class.
- writeFile which was used to write the output of generateConnections to a text file.
- writeResult which was used to write out the output of the Graph class' main method and process request methods to the Data and Result text files respectively.

The Graph class used the Edge, Vertex, GraphException, and Path classes to construct the graphs given by the text files consisting of edges generated by the generateConnecitons method.

I also edited the Graph class to allow for the counting of the number of:

- Vertices processed
- Edges processed
- Operations regarding the priority queue

In addition to this, I changed the main method so that the program no longer required user input and it would conduct a series of tests (20 tests) for each of the 25 graphs. I also modified the Graph class to call the methods in the GraphGenerator class to write the output of those tests into a text file called Data.txt and the data used in this experiment into a text file called Results.txt.

I used textfiles to store the grapes and the results of the experiment and I also used a Priority queue to store the nodes that have yet to be processed when executing the Dijkstra's algorithm.

Experiment Aim and Design

Aim:

The aim of this experiment was to compare the practical performance of the Dijkstra's algorithm to its theoretical time complexity of O(|E|log|V|) where |E| is the number of edges in the graph, and |V| is the number of vertices in the graph to determine if these were sufficient performance upper bounds.

Design:

The execution of this experiment had 25 different graphs, each with a different combination of vertices and edges. These 25 graphs were grouped into 5 groups where each graph had the same number of vertices in the same group, but a different number of edges.
The way that I designed the graphs was such that they used 2 letters to serve as the nodes which were selected randomly, and a random integer from 1-10 to serve as the weight of the edge.
The issue with this choice is that each graph was only limited to a maximum of 26 nodes, and a maximum of 676 edges – I didn't allow for edges to be repeated.

I ran each graph 20 times using randomly generated inputs and recorded the results in Data.txt. I then recorded the average of these graphs counts and stored the values in Results.txt, which included:

- The number of vertices in the graph
- The number of edges in the graph
- The number of vertices that were processed
- The number of edges that were processed
- The number of operations used by the priority queue

To ensure that every graph would behave like a graph with the same number of vertices, I implemented an algorithm to ensure that every vertex would be connected to the graph in some way
**see creativity section for more detail*

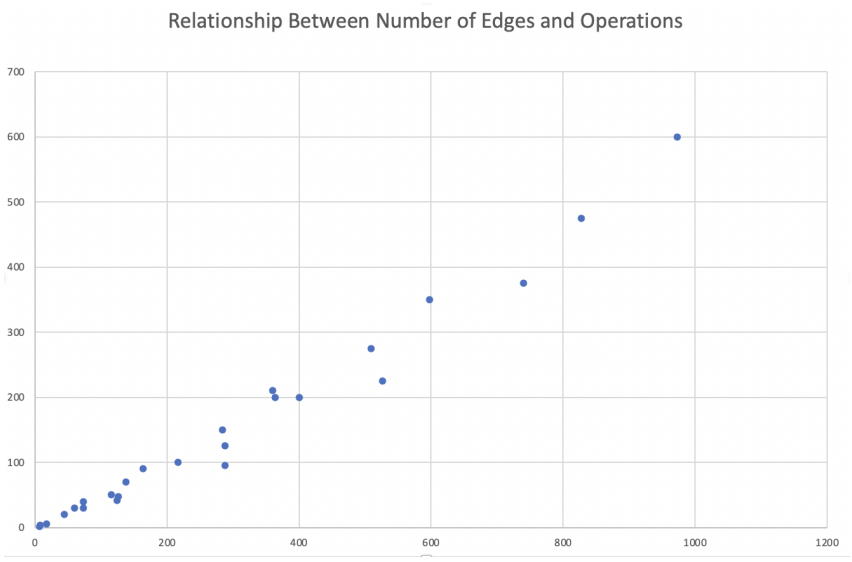
Experiment Results

The following results were obtained by the experiment.

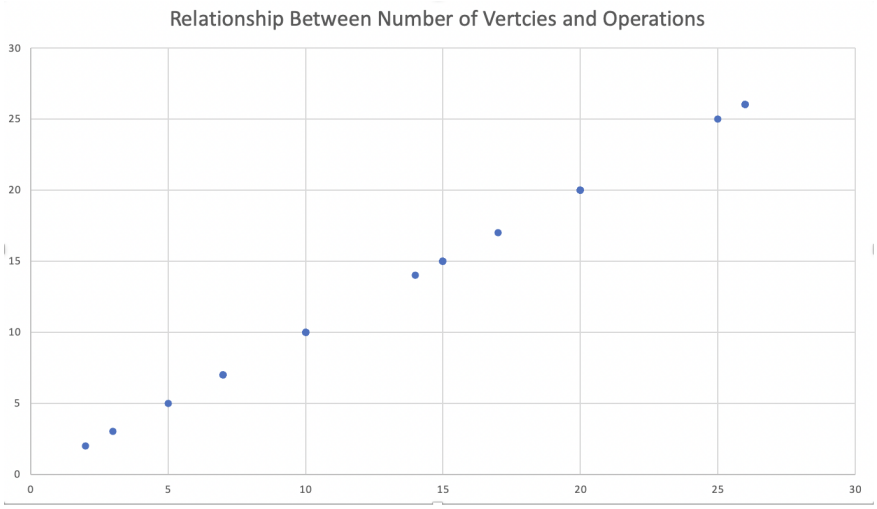
Vertices	Edges	VCount	ECount	PQCount	Number of Operations	O(|E|log(|V|))
7	10	3	4	1	8	28,07354922
7	20	7	20	17	44	56,14709844
7	30	7	30	23	60	84,22064766
7	40	7	40	26	73	112,2941969
10	10	2	2	3	7	33,21928095
10	30	10	30	33	73	99,65784285
10	50	10	50	55	115	166,0964047
10	70	10	70	58	138	232,5349666
10	90	10	90	64	164	298,9735285
15	20	5	5	7	17	78,13781191
15	50	14	47	65	126	195,3445298
15	100	15	100	102	217	390,6890596
15	150	15	150	119	284	586,0335893
15	200	15	200	149	364	781,3781191
15	210	15	210	135	360	820,4470251
20	50	17	42	65	124	216,0964047
20	125	20	125	143	288	540,2410119
20	200	20	200	180	400	864,385619
20	275	20	275	214	509	1188,530226
20	350	20	350	228	598	1512,674833
26	100	25	95	168	288	470,0439718
26	225	26	225	275	526	1057,598937
26	375	26	375	339	740	1762,664894
26	475	26	475	327	828	2232,708866
26	600	26	600	347	973	2820,263831

***Note that VCount, ECount and PQCount are the number of operations executed in the experiments**
***Number of Operations is the sum of the 3 counts mentioned above**

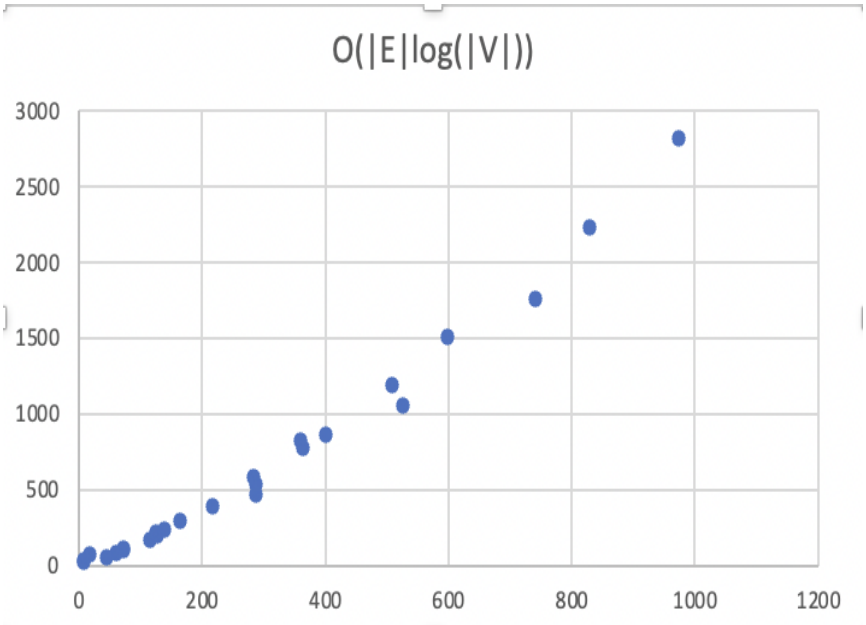
This data was used to plot the following graphs:



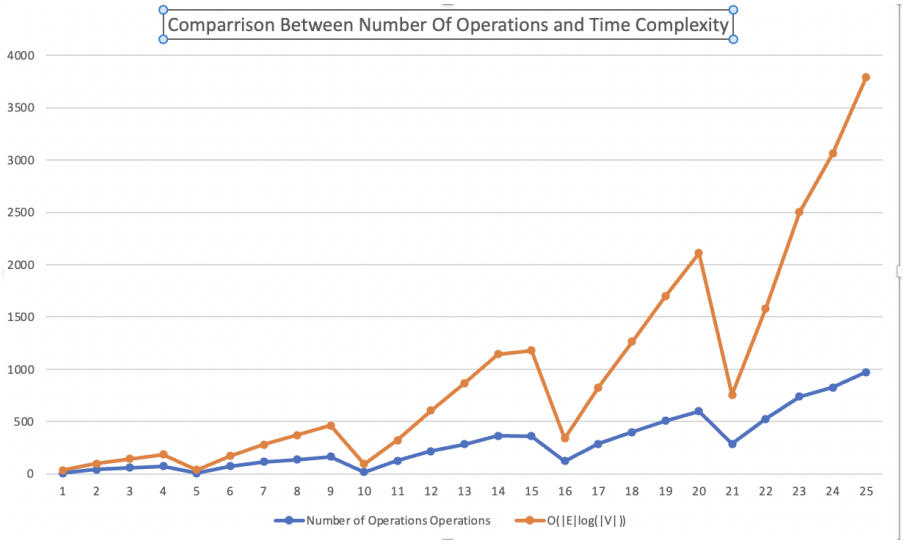
Where Number of Edges is on the Y-axis and Number of Operations is on the X-axis



Where Number of Vertices is on the Y-axis and Number of Operations is on the X-axis



Where Number of Operations is on the Y-axis and Number of Operations is on the X-axis



From these graphs, we can conclude that there is a:

- Strong positive linear relationship between the Number of edges and number of operations
- Strong positive linear relationship between the Number of vertices and number of operations
- Moderate positive exponential relationship between the Number of operations and time complexity

We can also conclude from the final graph that the time complexity for Dijkstra's algorithm of O(|E|*log(|V|)) is a sufficient theoretical upper bound for the algorithm as the blue line representing the practical trials never crossed the orange line representing the time complexity.

Creativity

The creativity for this project came in the form of data generation of the graphs.

To ensure that the graphs behaved like they were graphs that had the correct number of vertices, I implemented an algorithm to ensure that every vertex was connected to at least 1 other vertex. To do this, I incremented the start vertex by 1 every third edge generated but I incremented the destination vertex by 1 every time. This resulted in graph files with the pattern:

AB; AC; AD; BE; BF; BG; CH etc... ensuring that every node would be connected in the graph.

I also applied various formatting to ensure that the data generated was more readable such as:

- I added 3 lines at the end of every graph file to note the number of edges and vertices in the graph
- I added line breaks and empty lines in between each graph experiment to segment the graphs
- I added a heading above each graph's set of results consisting of the graph's name as well as the number of vertices and edges that it possessed.

In addition to this, I ensured that my experiment would be more accurate by applying more data in the form of using the average of 20 experiments to minimise the effects of outliers.

Git Usage Log

0: commit 7f5b81cee9eaee1a297f4c57cbe8e70e3166f7fe

1: Author: Christopher Blignaut <\u00a8chriss3rd@gmail.com\u00a8>

2: Date: Fri May 5 16:31:07 2023 +0000

3:

4: Fixed the damn Makefile

5:

6: commit e22c429cbdbce58a935fd617648d88d3e55a6b99

7: Author: Christopher Blignaut <\u00a8chriss3rd@gmail.com\u00a8>

8: Date: Fri May 5 16:21:42 2023 +0000

9:

...

37: Author: Christopher Blignaut <\u00a8chriss3rd@gmail.com\u00a8>

38: Date: Fri Apr 28 19:18:33 2023 +0000

39:

40: Added non-working Makefile

41:

42: commit 5302e3eb762a60cb3cbaac2575988b1b3297fdab

43: Author: Christopher Blignaut <\u00a8chriss3rd@gmail.com\u00a8>

44: Date: Fri Apr 28 18:53:58 2023 +0000

45:

46: Added project to git