Thabang Mokoena(MKNTHA093)
Dijkstra Experiment

OOP Design:

To start of the experiment I designed a GraphGenerator class which generates file text containing edges of the graphs so I can could access the files and run Diskstra’s algorithm on each graph generated.

I used the path class which implements the comparable interface to store the paths in the graph which would be used in the implementation of the priority queue used in Dijkstras algorithm. The edge class stores the destination and the weight of that edge and is used by the vertex class to store adjacent nodes. The vertex class provides a way to represent and store information about the vertices of the graph. Graph class contains a `Map<String, Vertex>` called `vertexMap`, which maps vertex names to their corresponding Vertex objects , it also contains Dikstra’s algorithm along with a read method which reads edges from the graph files and adds them to the g and then after run the algorithm on the graph.

Finally I implemented the experiement class which basically runs through all the graph files with the help of the read the read function and stores the results of each graph in a results file in the following format(vertex, edges, ElogV, operations).

Goal:

The objective of the analysis is to compare the actual performance of Dijkstra's shortest paths algorithm with its theoretical bounds in terms of the number of comparisons that are made in the algorithm. Dijkstra's calculation is ensured to find the shortest way between a source vertex and any remaining vertices in a weighted chart with non-negative edge costs, but its performance can change contingent upon the size and design of the graph. Hypothetical limits provide us with an worst-case time and space complexity of the algorithm, however genuine execution can be impacted by different variables. By counting the times examinations are made in a given diagram, By counting the number of times comparisons are made in a given graph, we can compare the actual performance of the algorithm with its theoretical bounds and gain insight into how the algorithm behaves in practice.

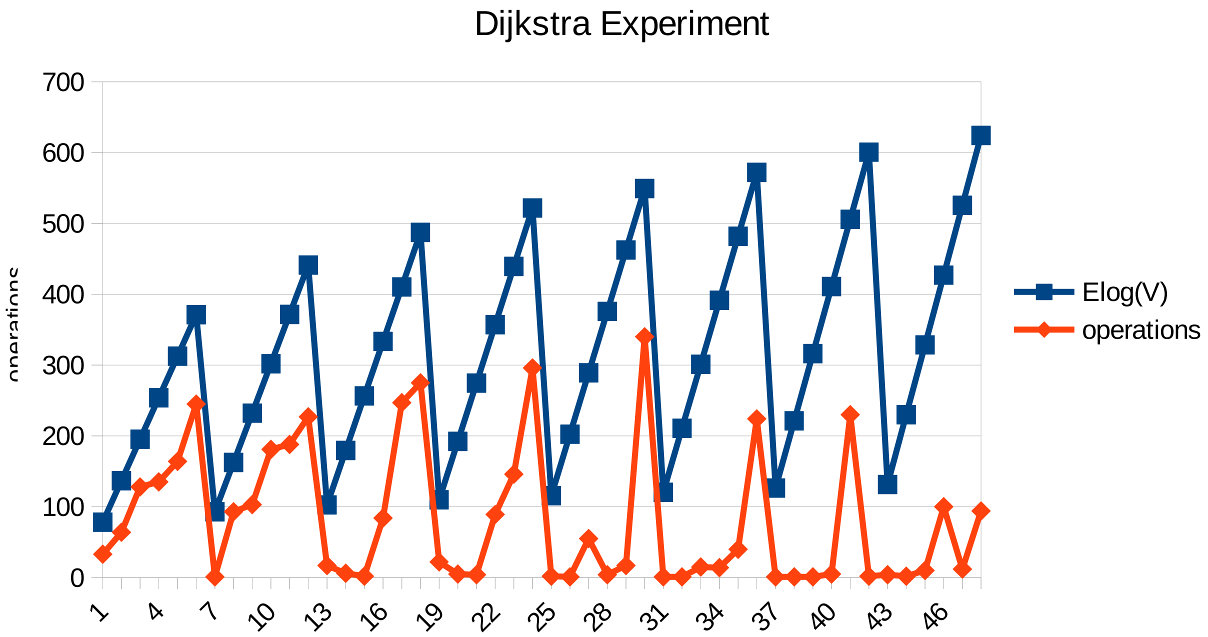
Execution:

To execute the experiment i choose set of graphs of varying sizes and structures. Then i added instrumentation on the algorithm to count the number of comparisons done in the algorithm for each graph. I then generated test cases on the randomly created graphs
of varying size and structures using Dijkstra's algorithm then i recorded the number of comparisons done in each graphs. I also calculated the theoretical bounds for each given graph pf varing vertices and edges. I then went on to compare the theoritical bound with my counted number of operations and recorded the results in a table form and graph form.

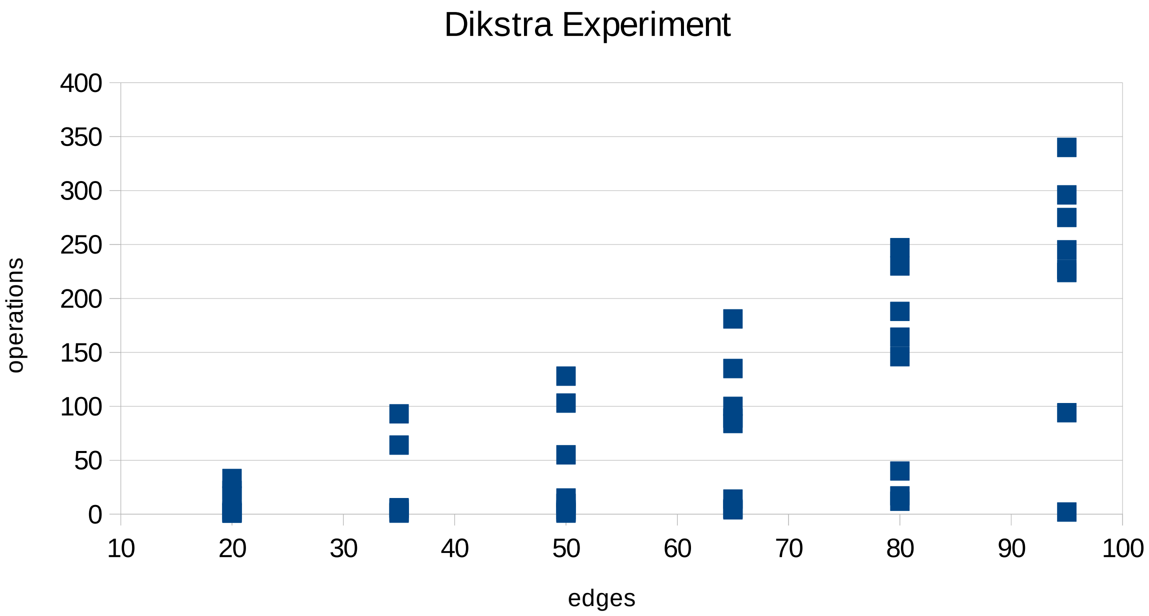
Results:

Vertexes	Edges	Elog(V)	operations
15	20	78,13781	33
15	35	136,74117	64
15	50	195,34453	128
15	65	253,94789	135
15	80	312,55125	164
15	95	371,15461	245
25	20	92,87712	1
25	35	162,53497	93
25	50	232,19281	103
25	65	301,85065	181
25	80	371,5085	188
25	95	441,16634	227
35	20	102,58566	17
35	35	179,52491	6
35	50	256,46415	2
35	65	333,4034	84
35	80	410,34264	247
35	95	487,28189	275
45	20	109,83706	22
45	35	192,21486	5
45	50	274,59265	4
45	65	356,97045	89
45	80	439,34825	146
45	95	521,72604	296
55	20	115,62719	2
55	35	202,34759	1
55	50	289,06799	55
55	65	375,78838	4
55	80	462,50878	17
55	95	549,22917	340
65	20	120,44736	1
65	35	210,78287	1
65	50	301,11839	15
65	65	391,45391	14
65	80	481,78943	40
65	95	572,12494	224
80	20	126,43856	1
80	35	221,26748	1
80	50	316,0964	1
80	65	410,92533	5
80	80	505,75425	230
80	95	600,58317	2
95	20	131,39711	4
95	35	229,94495	2
95	50	328,49278	10
95	65	427,04061	100
95	80	525,58845	12
95	95	624,13628	94

GRAPH1



GRAPH2



Discussion:

The following graph1 provides insights into the algorithm's actual performance in comparison with its theoretical bounds. We can generally observe that even though the number of operations increase its still less than the theoretical bound which confirms that for a worst case scenerio the time complexity will still be less than ElogV which is the theoretical bound.

From the provided results(Graph2), we can observe that the number of operations (which is the number of comparisons done) increases as the size of the graph increases. This is expected since larger graphs have more nodes and edges that need to be processed to calculate the shortest path between two nodes.

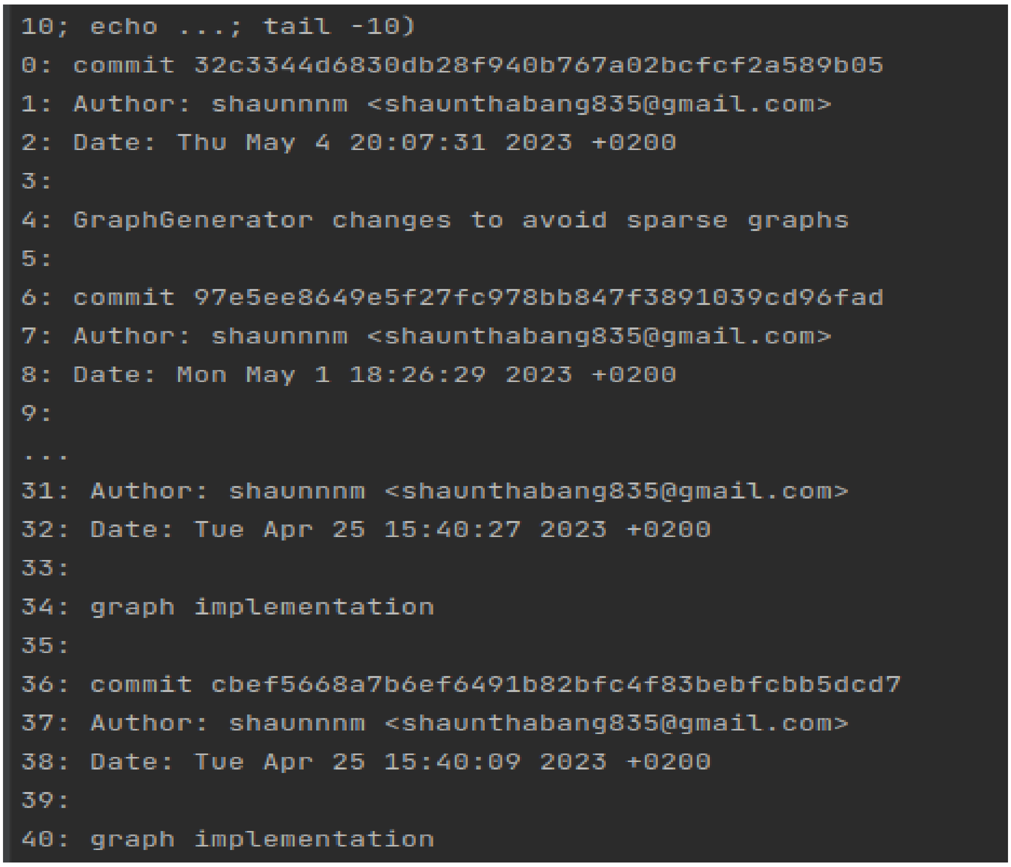
However, the rate of increase varies depending on the density of the graph, as indicated by the Elog(V) value. For example we can see that for 35 vertices and 20 edges 17 operations are performed but for 35 vertices and 35 edges the number of operations decrease to 6 .This suggests that the running time of the algorithm is affected not only by the size of the input, but also by the structure of the input graph ,the starting node and other factors.

Overall, the data provides some insight into the performance of the algorithm for different input sizes, and highlights the importance of understanding the properties of the input data in order to optimize the performance of graph algorithms.

Creativity:

Instead of just counting the number of vertices I chose to count the number of operations as counting the number of comparisons and the number of times the priority queue is used can provide more detailed information about the performance of the algorithm than simply counting the number of vertices and edges processed. This is because the number of comparisons and priority queue operations directly impact the time complexity of the algorithm.

Git:



```
10; echo ...; tail -10)
0: commit 32c3344d6830db28f940b767a02bcfcf2a589b05
1: Author: shaunnnm <shaunthabang835@gmail.com>
2: Date: Thu May 4 20:07:31 2023 +0200
3:
4: GraphGenerator changes to avoid sparse graphs
5:
6: commit 97e5ee8649e5f27fc978bb847f3891039cd96fad
7: Author: shaunnnm <shaunthabang835@gmail.com>
8: Date: Mon May 1 18:26:29 2023 +0200
9:
...
31: Author: shaunnnm <shaunthabang835@gmail.com>
32: Date: Tue Apr 25 15:40:27 2023 +0200
33:
34: graph implementation
35:
36: commit cbef5668a7b6ef6491b82bfc4f83bebfcbb5dcd7
37: Author: shaunnnm <shaunthabang835@gmail.com>
38: Date: Tue Apr 25 15:40:09 2023 +0200
39:
40: graph implementation
```