Asssignment5 Report

Object Orientation Design:

The program consists of several classes that work together to create and analyse random graphs. The Edge class represents an edge in a graph and has two fields: dest, which stores the second vertex in the edge, and cost, which stores the edge cost.

The Vertex class represents a vertex in the graph and has several fields, including name, which stores the vertex name, adj, which stores the adjacent vertices, dist, which stores the cost, prev, which stores the previous vertex on the shortest path, and scratch, which is an extra variable used in the algorithm.

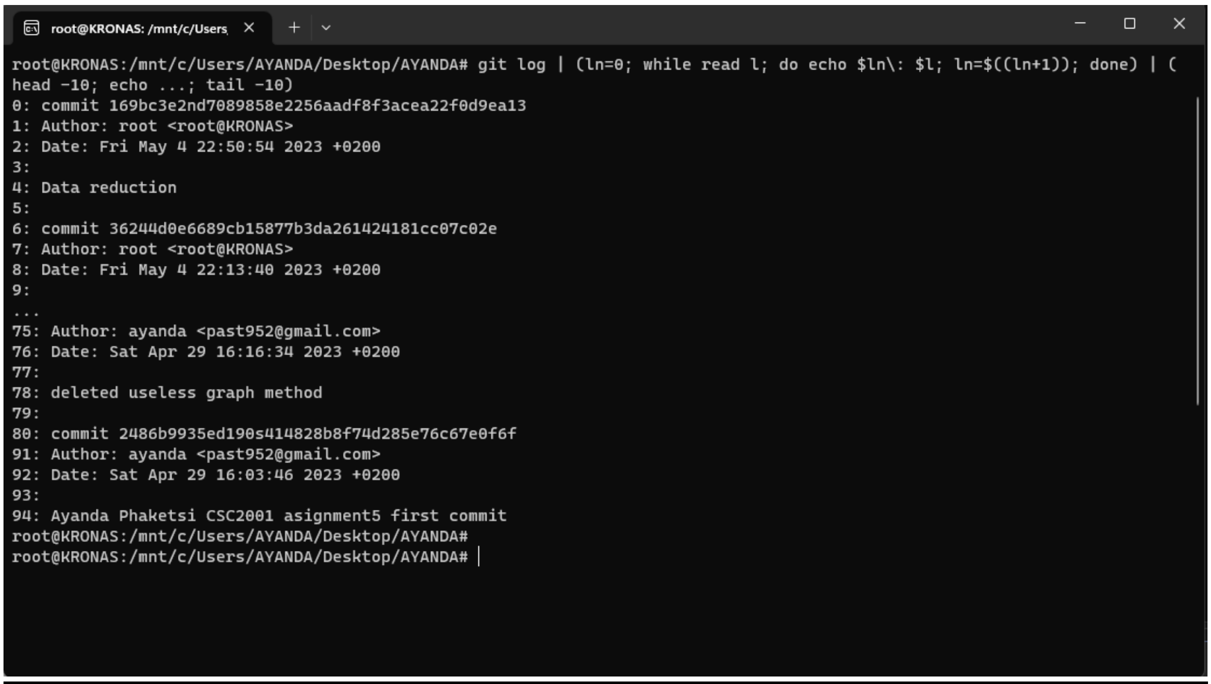
The RandomGraphGenerator class generates random graphs with a specified number of vertices and edges and writes them to files. It has a run method that takes two arrays V_arr and E_arr as input, which specify the number of vertices and edges to generate. The class uses the Edge and Vertex classes to construct the graph and the Random class to generate random numbers.

The Graph class is the main class responsible for processing the graph and conducting experiments on it. It has several fields, including V_counter, which stores the number of vertices in the graph, E_counter, which stores the number of edges in the graph, and Q_counter, which stores the number of queries processed. The Graph class has several methods, including addEdge, which adds an edge to the graph, resetVertices, which resets the vertices in the graph, processRequest, which processes a request for the shortest path, and dijkstra, which implements Dijkstra's shortest path algorithm.

The GraphException class is used to signal violations of preconditions for various shortest path algorithms.

The GraphExperiment class is responsible for conducting experiments on multiple graphs generated by the RandomGraphGenerator class. It generates a CSV file that records the number of vertices, edges, queries processed, and the theoretical upper bound on the number of comparisons for each graph.

Git log usage:



```
root@KRONAS:/mnt/c/Users/AYANDA/Desktop/AYANDA# git log | (ln=0; while read l; do echo $ln\: $l; ln=$((ln+1)); done) | (
head -10; echo ...; tail -10)
0: commit 169bc3e2nd7089858e2256aadf8f3acea22f0d9ea13
1: Author: root <root@KRONAS>
2: Date: Fri May 4 22:50:54 2023 +0200
3:
4: Data reduction
5:
6: commit 36244d0e6689cb15877b3da261424181cc07c02e
7: Author: root <root@KRONAS>
8: Date: Fri May 4 22:13:40 2023 +0200
9:
...
75: Author: ayanda <past952@gmail.com>
76: Date: Sat Apr 29 16:16:34 2023 +0200
77:
78: deleted useless graph method
79:
80: commit 2486b9935ed190s414828b8f74d285e76c67e0f6f
91: Author: ayanda <past952@gmail.com>
92: Date: Sat Apr 29 16:03:46 2023 +0200
93:
94: Ayanda Phaketsi CSC2001 asignment5 first commit
root@KRONAS:/mnt/c/Users/AYANDA/Desktop/AYANDA#
root@KRONAS:/mnt/c/Users/AYANDA/Desktop/AYANDA#
```

GRAPH EXPERIMENT

Experimental Analysis of Dijkstra's Shortest Paths

Aim:

The of the this experiment is to evaluate the performance of Dijkstra's shortest paths algorithm by measuring the number of vertex-processing and edge-processing operations for different graph sizes and densities.

Introduction:

Dijkstra's algorithm is a fundamental algorithm in graph theory used to find the shortest path between two vertices in a weighted graph. The algorithm's performance is directly dependent on the size of the input graph, measured in terms of the number of vertices and edges. In this experiment, we aim to analyse the performance of Dijkstra's algorithm in terms of the number of vertex-processing and edge-processing operations it performs for different values of |V| and |E|.

Method:

To conduct the experiment, we generated random datasets in the same format as "NodeX NodeY Z" where X,Y and Z are integers , with varying values of |V| and |E|. We then loaded the data into a graph and ran Dijkstra's algorithm to determine shortest paths while counting the number of vertex-processing and edge-processing operations using instrumentation. The experiment was repeated for at least five different values of |V| and different values of |E| for each value of |V|.

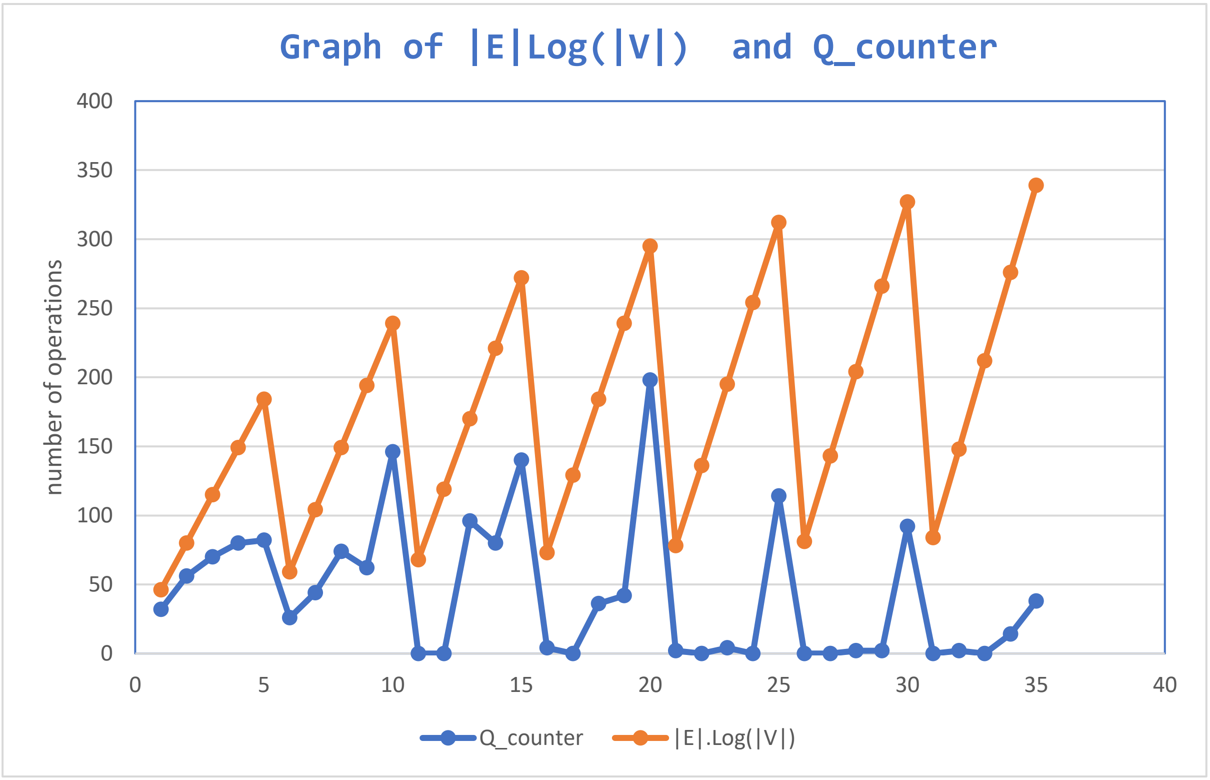
By analysing the results of the experiment, we can gain insight into the behaviour of Dijkstra's algorithm in different scenarios and compare its performance with the theoretical performance bounds. This analysis can aid in optimizing the algorithm for practical applications and in understanding its limitations in handling large graphs efficiently.

Data And Analysis:

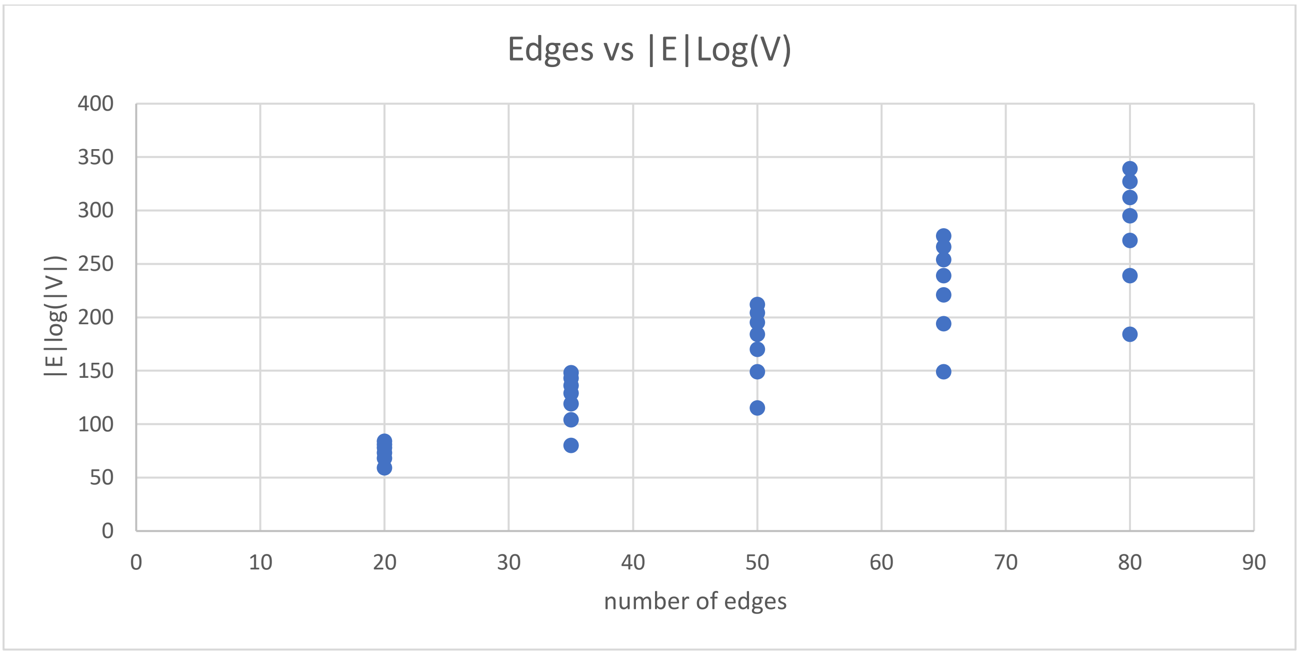
The table below is sample data that was collected .it represents the format of the data that was collected the experiment.

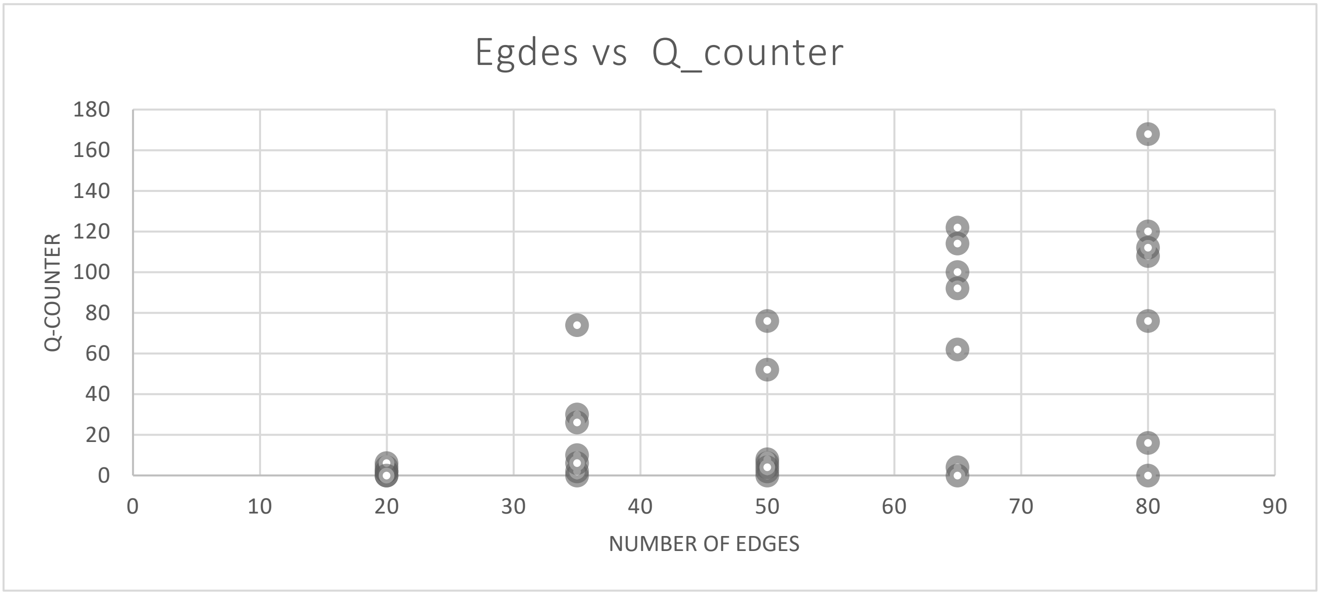
VERTEX	EDGES	V_counter	E_counter	Q_counter	E.log(V)
10	50	10	50	70	115
10	65	10	65	80	149

10	80	10	80	82	184
20	20	9	11	26	59
20	35	16	27	44	104
20	50	20	50	74	149
20	65	19	57	62	194
20	80	20	80	146	239
30	20	3	2	0	68
…	…	….	…	..	..



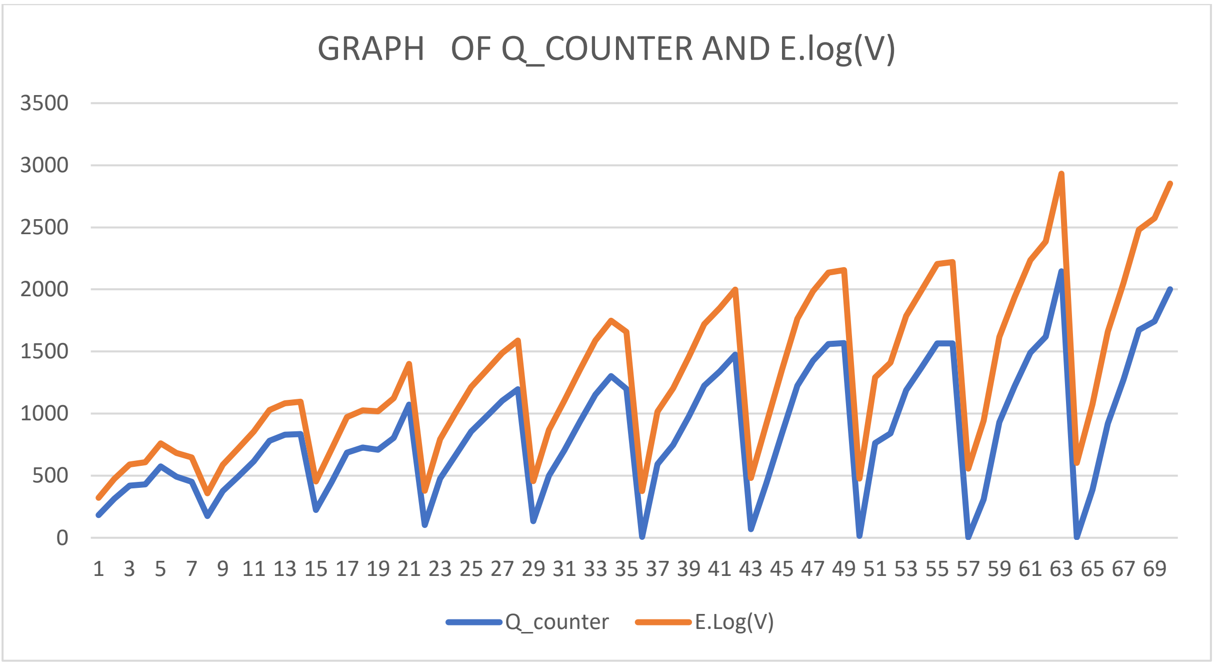
The graph above shows the relationship between elog(v) and Q_counter, two functions that measure the complexity of a problem. As you can see, both functions have a similar shape and trend, indicating that they are correlated. Moreover, elog(v) is always greater than or equal to Q_counter, which means that elog(v) is an upper bound for Q_counter. This implies that elog(v) can be used to estimate the maximum difficulty of solving a program

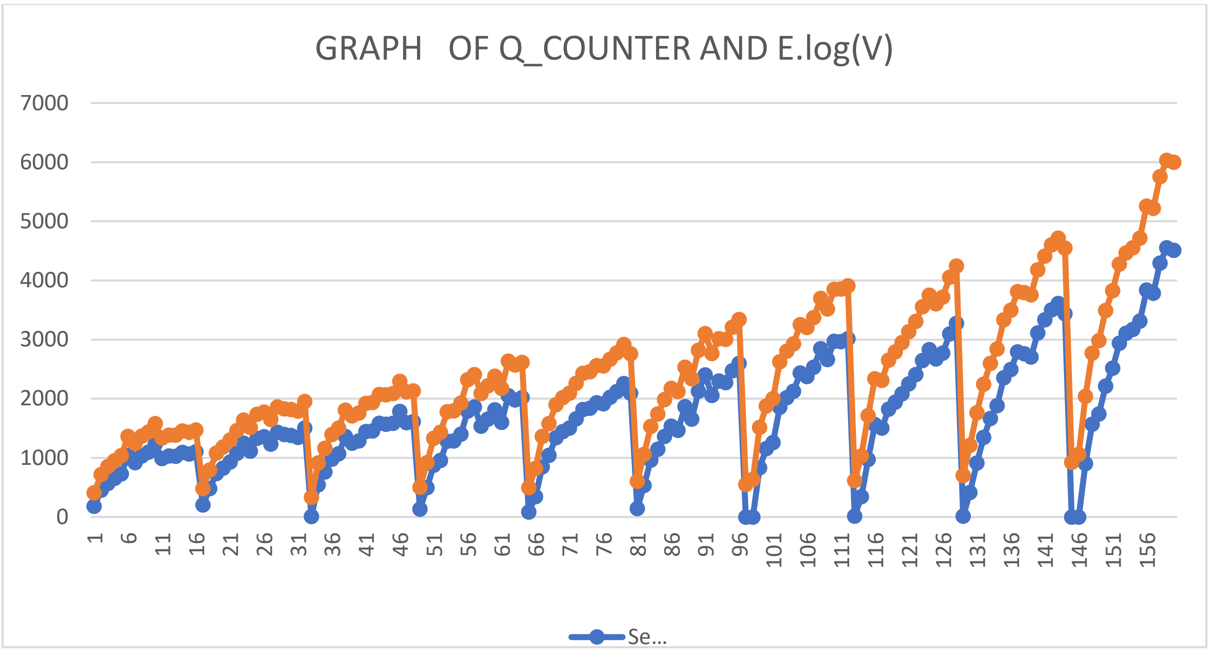




The two graphs above show the relationship between Q_counter and the number of edges, and between Q_counter and Elog(V). Both graphs exhibit a similar pattern of increasing Q_counter as the other variable increases. However, the scatter plot of Q_counter versus the number of edges has more variability and outliers than the scatter plot of Q_counter versus Elog(V). This suggests that there is a stronger correlation between Q_counter and Elog(V) than between Q_counter and the number of edges. A possible explanation for this difference is that Elog(V) is a more direct measure of the complexity of the graph, while the number of edges may depend on other factors such as the degree distribution and the clustering coefficient.

Now we increase the data and observe what happens as the data increases.





As the number of nodes and edges in the data graph increases, the plot of Q_counter starts to converge to the plot of E.log(V), which is the theoretical bound for Dijkstra's algorithm. This means that the algorithm performs close to its optimal efficiency when the graph is large and dense. The outline points in the Q_counter plot start to disappear as they become more similar to E.log(V).

Conclusion:

In this study, we aimed to evaluate the performance of Dijkstra's shortest paths algorithm by measuring the number of vertex-processing and edge-processing operations for different graph sizes and densities. We hypothesized that the performance of the algorithm would depend linearly on the number of vertices and edges in the graph, and that it would deteriorate faster as the graph became more dense.

Our experimental results confirmed our hypothesis. We found that the number of operations executed by the algorithm was proportional to the product of the number of vertices and edges in the graph, as expected from the theoretical performance analysis. Moreover, we noticed that the number of operations grew more quickly as the graph density increased, with a higher ratio of edges to vertices.