What your OO design is:

I firstly created 2 new classes called Data.java and GraphExperiment.java. My data class had a method to generate the dataset, in the correct format in a text file, by taking in the number of vertices and edges required for that specific dataset as parameters. It makes sure it achieves an equal number of edges, however due to the randomisation I applied, it is not guaranteed to generate the numVertices amount of unique vertices. I believe this is still ok as there are real life scenarios where some vertices are unreachable and this could make the graph more dense. The method makes sure no edge has the same source and destination, it ensures there are no duplicate edges and it assigns a random weight between 1 and 10 for all edges, to make sure there are no negative weights, which would cause Dijkstra’s algorithm to not work.

My graph experiment class makes use of this dataset generating class, by inputting each level of vertex and edge combination pairs. I did this by making 2 arrays, one for the vertex level values and another for the edge level values. I then ran a double for loop to ensure that for each vertex level, all 5 edge levels were made as datasets and recorded. After generating the dataset in a text file, it reads it into a graph, making use of the code provided by Hussein, but I changed it to a method to return a Graph object. For each test run, I record the instrumentation of applying Dijkstra's algorithm on the graph, ensuring I choose a node present in the graph. This then outputs this instrumentation into a .csv file, which I used later for interpreting and constructing graphs based on the instrumentation results.

In the graph class I wrote another function that takes in the dataset text file and returns a node name present in the dataset, to guarantee Dijkstra runs on a present node. I also added a function that takes the instrumentation results stored in the graph object, formats them separated by commas, and then writes the data to a csv file.

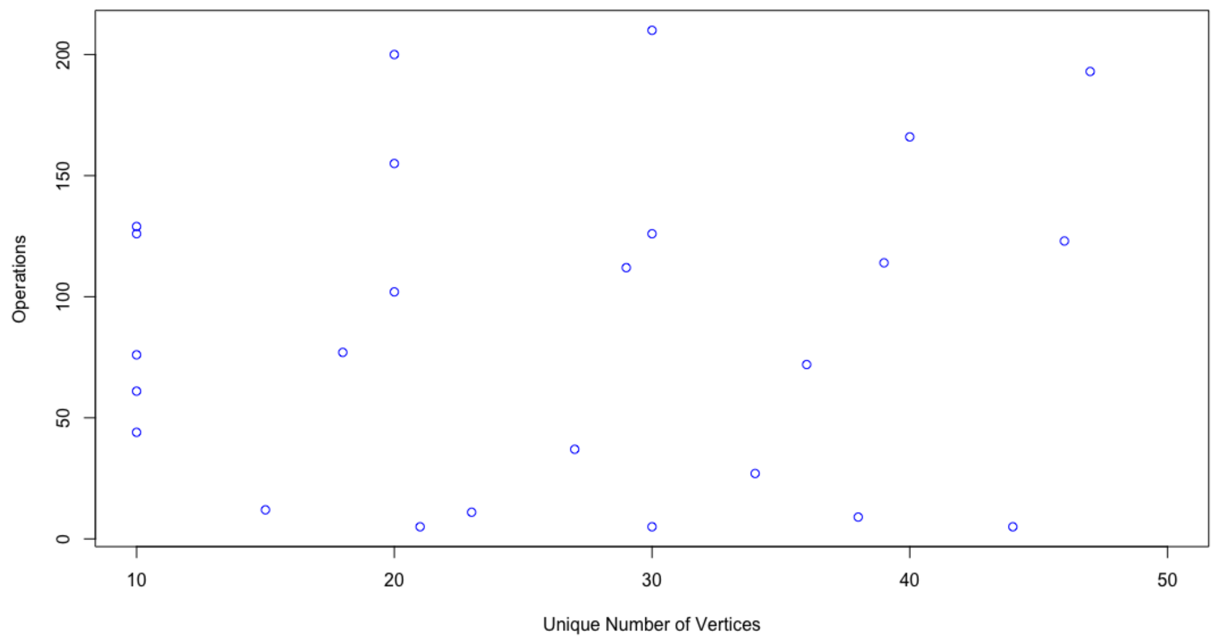
What the goal of the experiment is and how you executed the experiment:

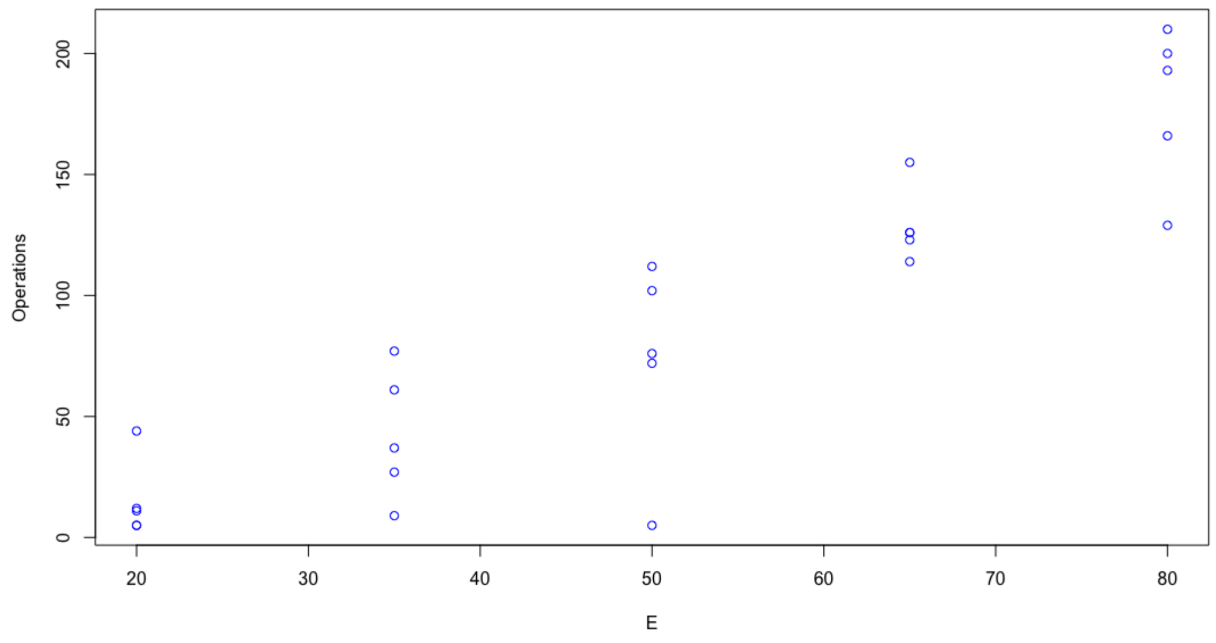
The ultimate goal of running this experiment was to to programmatically compare the performance of Dijkstra's shortest paths algorithm with the theoretical performance bounds.

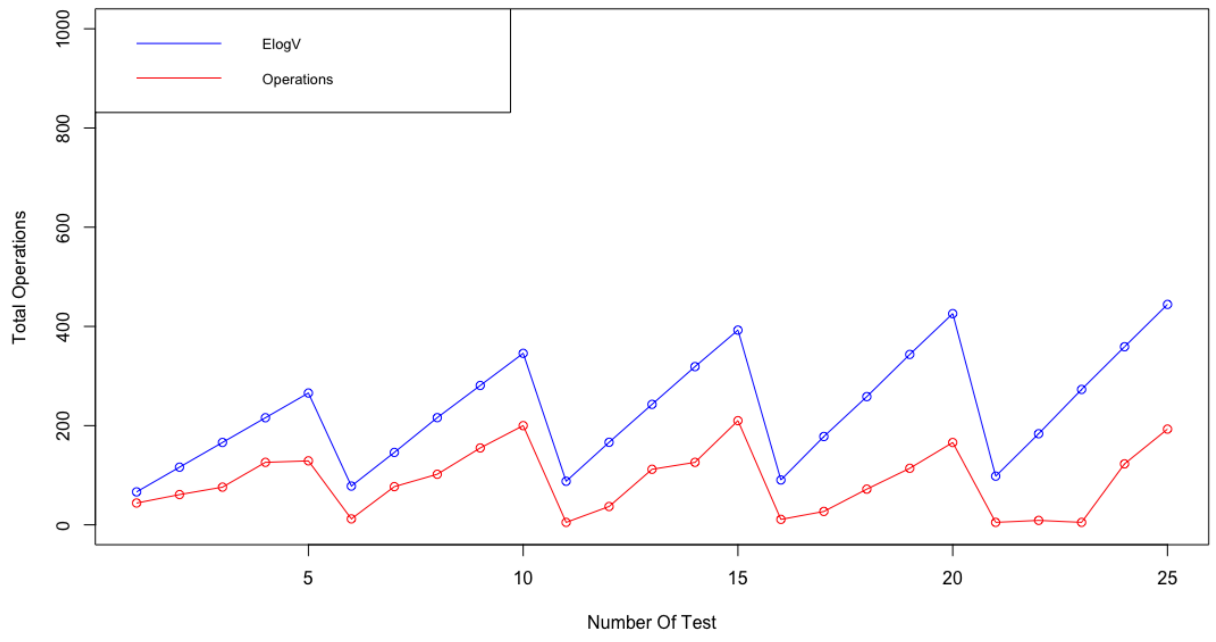
I chose to record the number of unique vertices,edges,vertex operation count, edge operation count,priority queue operation count, ElogV (Which is the time complexity of the algorithm) and the total number of operations which was calculated by adding vertex operations, edge operations and priority queue operations. I got these values by using instrumentation in the code, monitoring the performance. I applied the instrumentation in Dijkstra’s algorithm at the points where each operation was done.
Using these counts as data, I looked at how the total number of operations compared to the theoretical bounds, and whether it ever exceeded it or not and compared the number of edges and vertices to the number of operations.

For some cases of generating datasets, the number of unique vertices present was less than the number of inputted vertices into the dataset making function. The reason for this is the way edges were generated. A random source name and destination name were chosen from an array consisting of the max number of vertice names possible for that dataset and did not guarantee all possible vertices to occur in the dataset. This just means those missing vertices have a distance of infinity from the source and are not reachable. Nevertheless, the time complexity at different levels of vertices and edges was still always bounded by O(ElogV), and we can conclude that the results achieved are still significant and warranted further investigation.

Final Results/Graphs:







Graph Discussion:

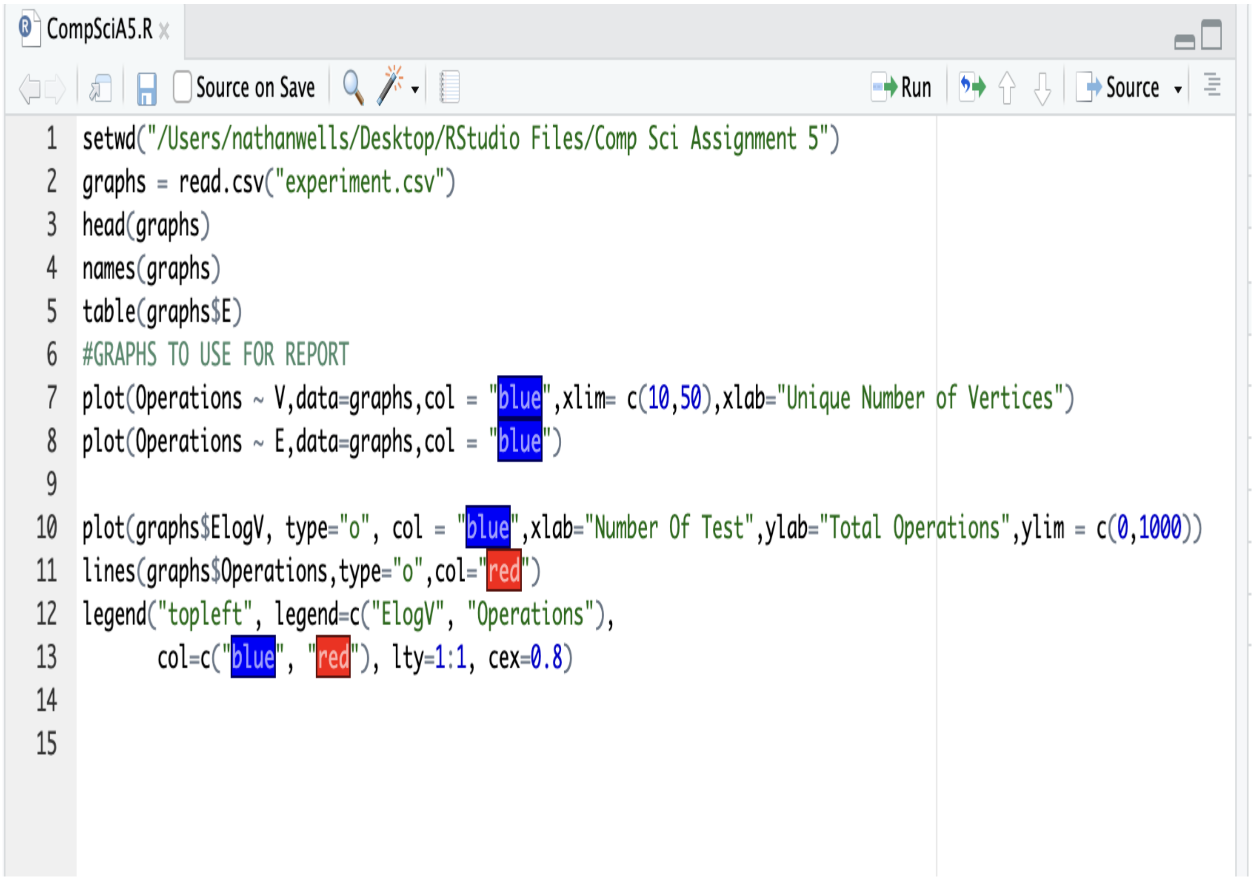
In the first graph, it is evident that there is no clear relationship between the unique number of vertices present and operations performed. This could be showing us that operations performed do not depend heavily on the number of unique vertices present.

However, in the second graph, it is clear that as the number of edges increases, so does the number of operations. This could be due to Dijkstra's algorithm having to do more operations as there are more edges to go down, aside from the one case where the operation count is very low for a high amount of edges.

It is clear from the third graph, that for each test we ran, the number of operations performed never exceeded ElogV, the theoretical bounds of dijkstra's algorithm. This confirms that the tests run were done correctly, never exceeding the maximum number of operations possible with Dijkstra's algorithm, which would cause confusion if it did exceed those bounds. We can also see that going in order of tests performed, that the tests with higher edge counts had more operations.

Creativity:

I used R to generate all of the graphs. Here is the code for generating the graphs above.

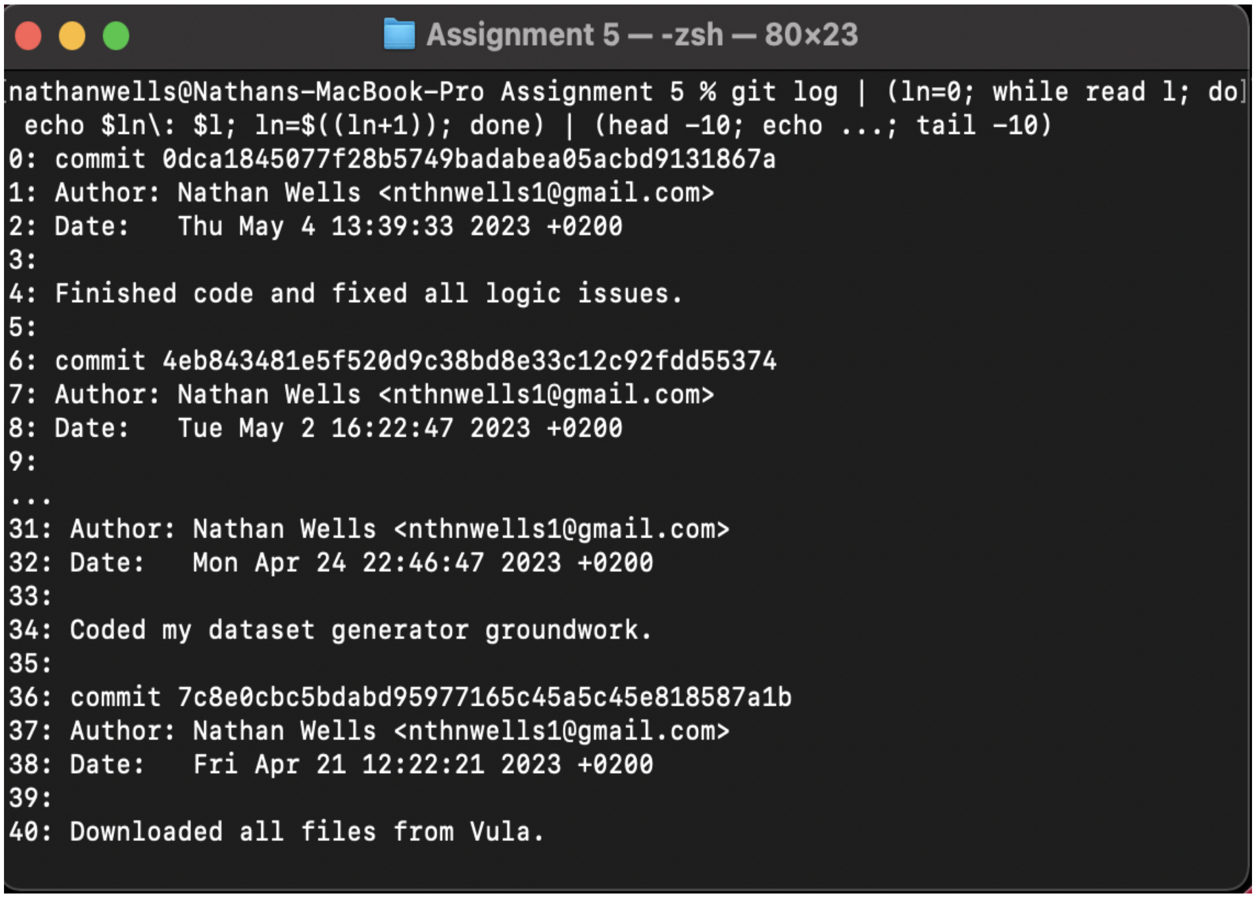


```r
setwd("/Users/nathanwells/Desktop/RStudio Files/Comp Sci Assignment 5")
graphs = read.csv("experiment.csv")
head(graphs)
names(graphs)
table(graphs$E)
#GRAPHS TO USE FOR REPORT
plot(Operations ~ V,data=graphs,col = "blue",xlim= c(10,50),xlab="Unique Number of Vertices")
plot(Operations ~ E,data=graphs,col = "blue")

plot(graphs$ElogV, type="o", col = "blue",xlab="Number Of Test",ylab="Total Operations",ylim = c(0,1000))
lines(graphs$Operations,type="o",col="red")
legend("topleft", legend=c("ElogV", "Operations"),
       col=c("blue", "red"), lty=1:1, cex=0.8)


```

Summary statistics from your use of git:



Assignment 5 — -zsh — 80×23

```
nathanwells@Nathans-MacBook-Pro Assignment 5 % git log | (ln=0; while read l; do
 echo $ln\: $l; ln=$((ln+1)); done) | (head -10; echo ...; tail -10)
0: commit 0dca1845077f28b5749badabea05acbd9131867a
1: Author: Nathan Wells <nthnwells1@gmail.com>
2: Date:   Thu May 4 13:39:33 2023 +0200
3:
4: Finished code and fixed all logic issues.
5:
6: commit 4eb843481e5f520d9c38bd8e33c12c92fdd55374
7: Author: Nathan Wells <nthnwells1@gmail.com>
8: Date:   Tue May 2 16:22:47 2023 +0200
9:
...
31: Author: Nathan Wells <nthnwells1@gmail.com>
32: Date:   Mon Apr 24 22:46:47 2023 +0200
33:
34: Coded my dataset generator groundwork.
35:
36: commit 7c8e0cbc5bdabd95977165c45a5c45e818587a1b
37: Author: Nathan Wells <nthnwells1@gmail.com>
38: Date:   Fri Apr 21 12:22:21 2023 +0200
39:
40: Downloaded all files from Vula.
```