CSC2009F ASSIGNMENT 5

Topic :Graphs

Name :Tendai Nyevedzanai

Student no:NYVTEN001

Date :29/03/2023

rite a report (of up to 8 pages) that includes the following:

- What your OO design is: what classes you created and how they interact (1 page at most).
- What the goal of the experiment is and how you executed the experiment. Report on any experiment design decisions you made.
- What your final results are (use one or more (mathematical) graphs), showing how the algorithm performs for different types/sizes of (data structure) graphs (different E and V values).
- Discuss what the results mean. Compare your results to the theoretical bounds.
- A statement of what you included in your submission that constitutes creativity - how you went beyond the basic requirements of the assignment.
- Summary statistics from your use of git to demonstrate usage. Print out the first 10 lines and last 10 lines from "git log" , with line numbers added. You can use a Unix command such as:

CLASSES CREATED

1. DataGenerator

   The Data Generator created the datasets. A File Writer function in java was used to write all the data generated into a text file. The data was generated using a for loop that iterated through the value of vertex we gave it be it for example 10 vertices or 20. The vertex number(int) and the string "vertex" were concatenated to give the vertex name. the first loop is for the first vertex and the inner loop is for the second vertex.

   Using the random function in java these two vertices are randomly joined together and a cost is added on the third column using the range input in random. All this data is then printed into the text file that was created .

2. GraphExperiment

   Firstly a File Reader function is used to read the text file. The text file under goes operations to be processed by methods in this class like splitting and indexing for easy manipulation of data. Various methods within the class allow for the data from the text file to be taken in as input and processed for example addEdge method that has first vertex as a parameter. The addEdge method makes use of Vertex objects to store the previous node and the Edge Class object to take in the cost. All this is is done to ouput a set of vertices and the cost.

   Within this class there is a Dijkstra method that finds the shortest path for the data generated, later used used to find operations.

   Having processed all the data in the text file and taking note of the eCount, vCount, pqCount the data is output in the main in the following arrangement: Vertex-Edge-eCount-vCount-pqCount.

3. Edge

   the Graph Experiment class. Simply takes the distance and cost. The object of the Edge class is used in the addEdge method of

4. Vertex Class

The object of the vertex class is used in the Graph Experiment class to take in the source name of vertex and the destination vertex. Which are used when randomly joining the two vertices. And Edge object is used to add the adjoining cost for the two.

5. Path Class

Use datatype of type Vertex to take the destination vertex, comparing it to the previous one.
Aim of Experiment

To use Dijkstra's shortest path algorithm to find the shortest path between two vertices using weighed edges.

Method

1. STEP ONE

Dataset Generation

Firstly I created a DataGenerator class that uses different ranges of vertices and edges but at fixed difference range to allow for proper analyses. For 10 vertices the cost of edges was varied from 20-80, this was done for 20,30,40 and 50 vertices. The increment of vertices was put at 10 to all for some uniformity. A for loop is used to create the vertices. The outer loop being for the first vertex and the inner loop for the second vertex. A random function was then used to place the cost of the edges between the two vertices

Having generated this data of two vertices against one cost and storing it in separate text files I moved to processing the data.

2. STEP TWO

Data Processing

I used a file reader to read from the text file. upon reading from it each line, since they are structured the same i split each line into separate indexed strings the first being the start vertex , second being the destination index and the third being the cost of the edge.

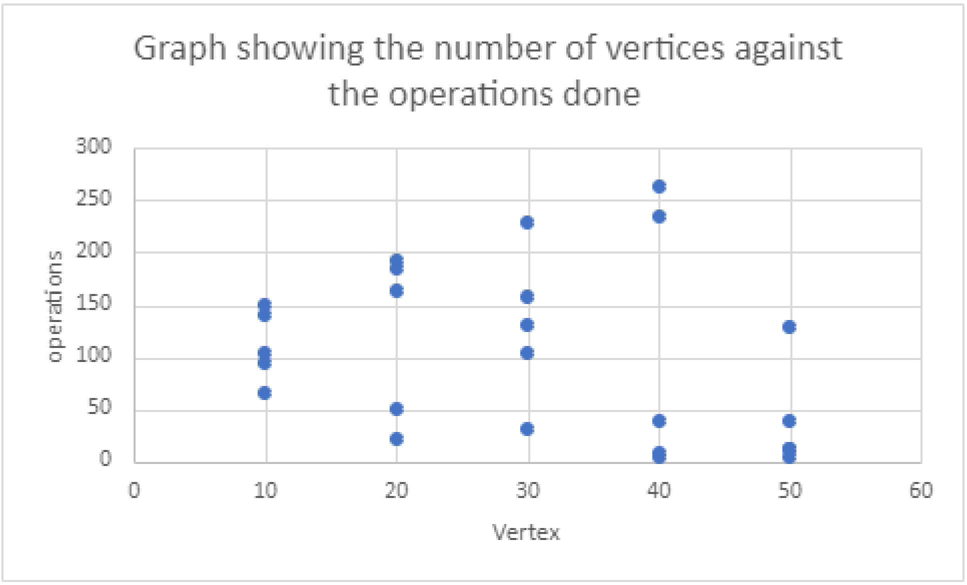
Having made variables for all these strings I placed them as parameters in the addEdge method and the Dijkstra's method using the GraphExperiment Class object. The Dijkstra's method finds the shortest path in the randomly generated datasets. The addEdge method takes note of the source vertex, destination vertex and the cost after finding the shortest path

3. STEP THREE

Instrumentation

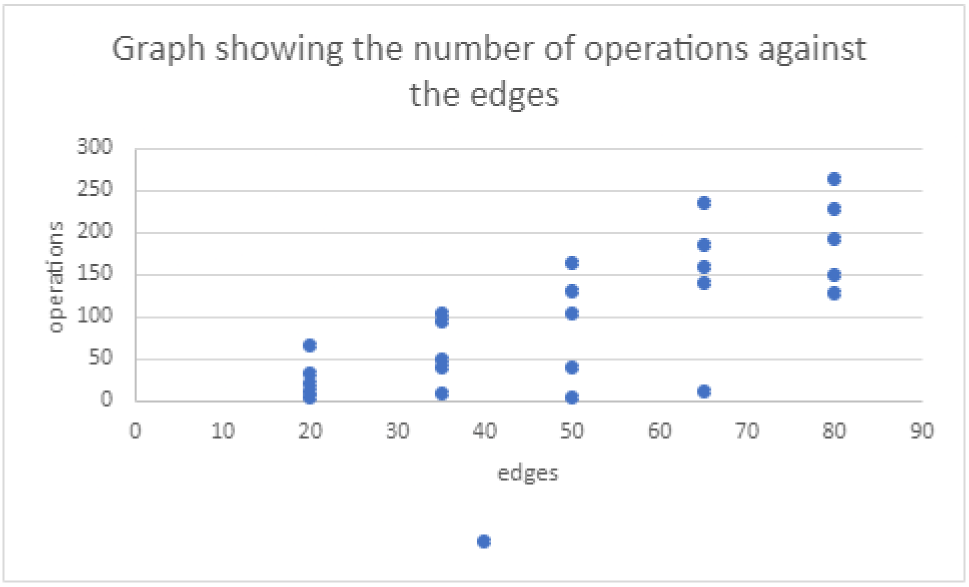
I added counters for edges, vertices and priority queue to find the number of operations, and to later calculate complexities . the eCount increments after every edge is processed , the vCount also increments after every Vertex processing . the pqCount increments every time a vertex is added into the priority queue. All these values are then printed against each pair of vertices and the respective cost between them. This is done for the analysis phase of experiment .

ANALYSIS



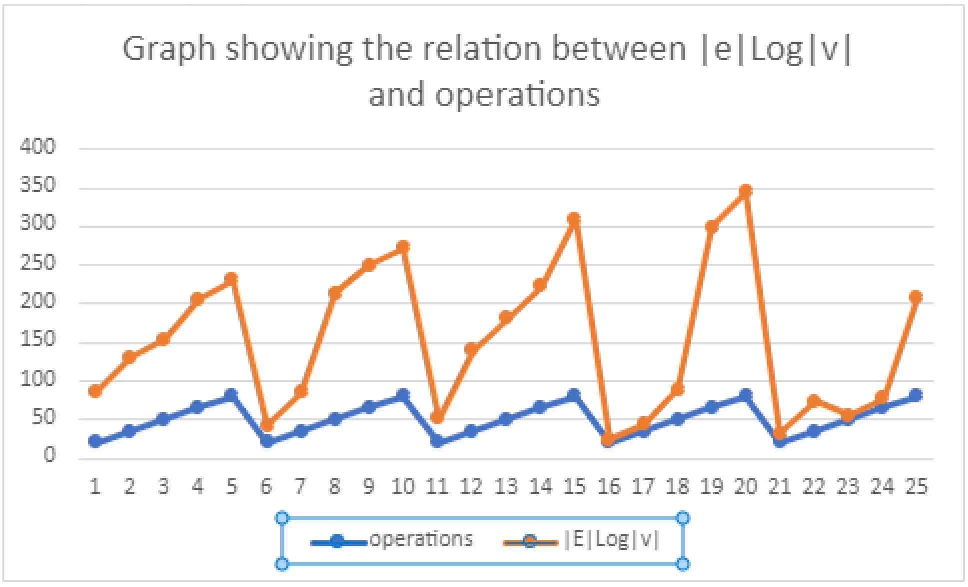
In the above graph as the number of vertices increases the range of operations appears to be increasing as well.

The number of vertices processed represents different a number of edges thus why our points are scattered. There appears to be a drop after 40 vertices which qualifies to say the operations do increase as the vertex number increases but only up to a certain number of vertices.



The above graph shows the number of operations undertaken against the number of edges . as the number of edges increases the bound of operations is also increasing.

The edges start of tight , but as they increase they begin to scatter also leading to the increase in operations



With the given data it appears that the number of operations are bound by |E|Log|V|, and given the closeness of the two trends , the operation trend id bound by a constant. Operations and E|Log|V|, are not related what so ever as E|Log|V| is a bound for the operations Undertaken

CONCLUSION

The results received programmatically do compare to the theoretical performance bounds and it is in fact that the complexity E|Log|V| is true for any number of vertices , given it is neither to sparse or too tight.

Git log

```
0: commit 2c8452f2efac1314fa5ea271a2b6f1d019704b75
1: Author: Tendai Nyevedzanai <nyvten001@nightmare.cs.uct.ac.za>
2: Date: Thu May 4 23:28:40 2023 +0000
3:
4: track
5:
6: commit 0617415049a1ee292523ccfdbc48481959083069
7: Author: Tendai Nyevedzanai <nyvten001@nightmare.cs.uct.ac.za>
8: Date: Thu May 4 16:56:26 2023 +0000
9:
...
19: Author: Tendai Nyevedzanai <nyvten001@nightmare.cs.uct.ac.za>
20: Date: Thu May 4 16:55:13 2023 +0000
21:
22: added the toString method
23:
24: commit b2ccd21abf66d67fdf00e4b09a09caf61afaa315
25: Author: Tendai Nyevedzanai <nyvten001@nightmare.cs.uct.ac.za>
26: Date: Sat Apr 29 15:07:46 2023 +0000
27:
28: created datafiles
```