Analysis document: GraphUtility

● Graded

Student

HARRY KIM

Total Points

30 / 30 pts

Question 1

Question 1

1 / 1 pt

✔ **– 0 pts** Correct

Question 2

Question 2

2 / 2 pts

✔ **– 0 pts** Correct

Question 3

Question 3

5 / 5 pts

✔ **– 0 pts** Correct

Question 4

Question 4

5 / 5 pts

✔ **– 0 pts** Correct

Question 5

Question 5

5 / 5 pts

✔ **– 0 pts** Correct

Question 6

Question 6

4 / 4 pts

✔ **+ 4 pts** Correct

Question 7

Question 7

4 / 4 pts

✔ **+ 4 pts** Correct

Question 8

Question 8

4 / 4 pts

✔ **+ 4 pts** Correct

Question 9

Late Penalty

0 / 0 pts

✔ **– 0 pts** submitted before deadline

Questions assigned to the following page: 1, 2, 3, and 4

Harry Kim
U1226472
10/21/2020
CS 2420

Analysis Document — Assignment 6

1. I am **not** invoking one of my three exemptions for pair programming due to an extenuating circumstance. My programming partner is Braden Morfin, and Braden is the one who submitted the program to Gradescope.

2. Pair programming experience:
    - My partner and I spent about 8 hour to complete the assignment and create tests for the assignment.
    - Braden's very smart and is a nice person, he's good at communication, and he codes very well. I would plan to work with Braden again.

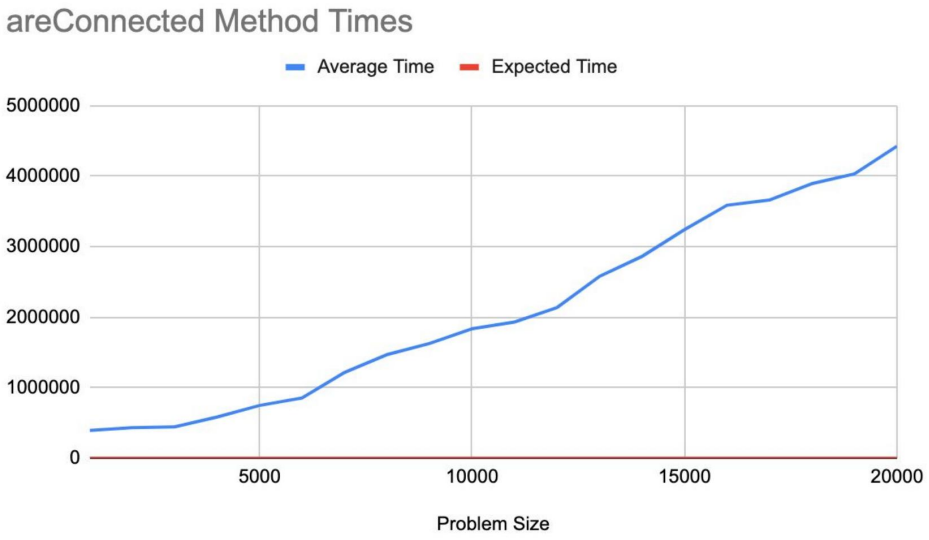
3. In order to assess the run-time efficiency of the depth-first search algorithm via the areConnected method, we first ran the test through a for loop which incremented the problem size to be used in each loop by sizes of 1,000 and went up to no greater than 20,000. We then used the *generateRandomDotFile* from our lab 7 which randomly constructed either a digraph with the size of the problem sizes. We created new ArrayLIsts of type String, one for sources and one for destinations, and a random source data value and a random destination data value, both within the parameters of the problem size, to pass it into our GraphUtility methods. To test the depth-first search, we simply called our areConnected method, with the correct parameters previously stated, within another for loop which ran 250 loops in order to ensure more accurate data. To time the *areConnected* method, we took the time it took to complete the for loop which tests the method and divided said time with the times to loop to get the average time. Using the "CheckAnalysis" technique, we also took the average time and divided it by the problem size because we expected this method to behave in a linear fashion of O(N).

4. In order to assess the run-time efficiency of the breadth-first search algorithm via the *shortestPath* method, we first ran the test through a for loop which incremented the problem size to be used in each loop by sizes of 1,000 and went up to no greater than 20,000. We then used the *generateRandomDotFile* from our lab 7 which randomly constructed a digraph with the size of the problem sizes. We created new ArrayLIsts of type String, one for sources and one for destinations, and a random source data value and a random destination data value, both within the parameters of the problem size, to pass it into our GraphUtility methods. To test the breadth-first search, we simply called our *shortestPath* method, with the correct parameters previously stated, within another for loop which ran 250 loops in order to ensure more accurate data, but this time we surrounded the method call with a try-catch to ensure that the code only times the

Questions assigned to the following page: 4, 5, and 6

method if there exists a path between the source data and the destination data and just continue if it catches the Illegal argument exception. To time the *shortestPath* method, we took the time it took to complete the for loop which tests the method and divided said time with the times to loop to get the average time. Using the "CheckAnalysis" technique, we also took the average time and divided it by the problem size because we expected this method to behave in a linear fashion of O(N).

5. In order to assess the run-time efficiency of the topological sort algorithm via the *sort* method, we first ran the test through a for loop which incremented the problem size to be used in each loop by sizes of 1,000 and went up to no greater than 20,000. We then used the *generateRandomDotFile* from our lab 7 which randomly constructed a digraph with the size of the problem sizes. We created new ArrayLIsts of type String, one for sources and one for destinations, to be topologically sorted. To test the topological sort algorithm, we simply called our *shortestPath* method, with the correct parameters previously stated, within another for loop which ran 250 loops in order to ensure more accurate data. To time the *shortestPath* method, we took the time it took to complete the for loop which tests the method and divided said time with the times to loop to get the average time. Using the "CheckAnalysis" technique, we also took the average time and divided it by the problem size because we expected this method to behave in a linear fashion of O(N).

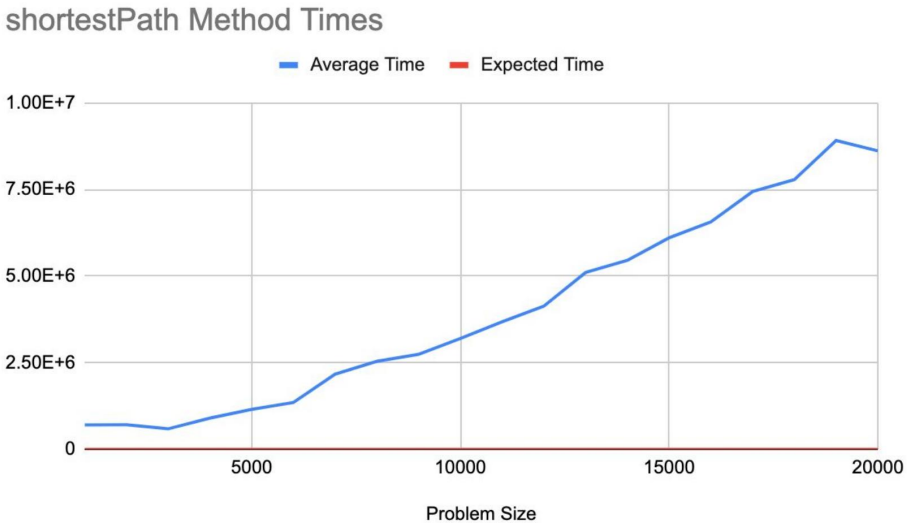


6.

As expected, the graph demonstrates that the *areConnected* method, and in turn the depth-first search algorithm, grows in a linear fashion. We also know that it is working properly since using the "CheckAnalysis" technique shows us that the expected time is converging to a positive number and thus is working properly. The relationship between |V| and |E| for the *areConnected* method does not affect the performance because it only

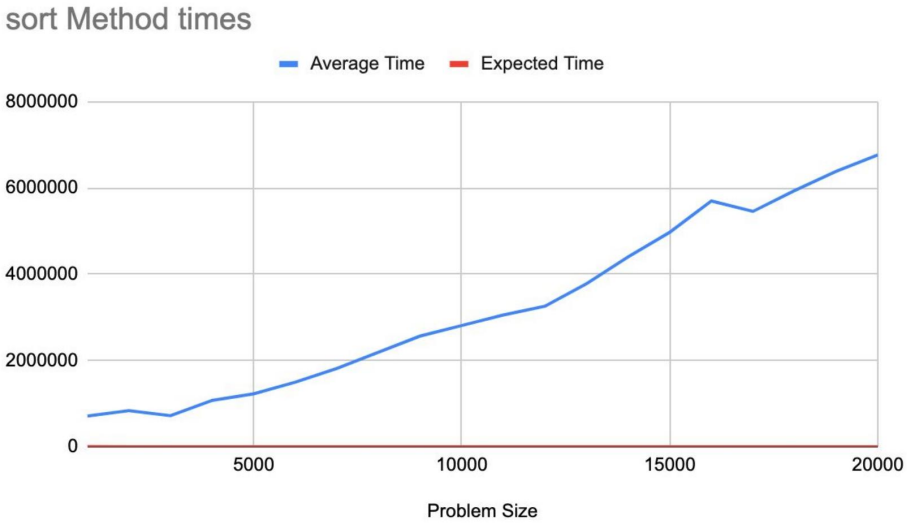
Questions assigned to the following page: 6 and 7

needs to find a single path between the source data and the destination data, and not down every path, regardless of how many paths there are to ensure that the two vertices are indeed connected.



7. As expected, the graph demonstrates that the *shortestPath* method, and in turn the breadth-first search algorithm, grows in a linear fashion. We also know that it is working properly since using the "CheckAnalysis" technique shows us that the expected time is converging to a positive number and thus is working properly. The relationship between |V| and |E| for the *shortestPath* method does affect the performance because it needs to find a single shortest path between the source data and the destination data by sequentially going down every edge connected to a vertex in order to find the shortest path to the destination data.

Question assigned to the following page: 8



8. As expected, the graph demonstrates that the *sort* method, and in turn the topological sort algorithm, grows in a linear fashion. We also know that it is working properly since using the "CheckAnalysis" technique shows us that the expected time is converging to a positive number and thus is working properly. The relationship between |V| and |E| for *sort* method does affect the performance because it needs to find and sort the vertices.