Collin Gros
11/09/2019
CS-372-M01

LAB 7

INTRODUCTION

Searching large graphs proves to be a monumental task when dealing with big data. Google Maps operates on a huge database, likely representing places as nodes and routes as edges. In a situation like that, an algorithm that is above O(n^2) is difficult to use, a map needs to be generated relatively quickly. DFS is useful for finding connected components, as well as sink/source nodes. BFS is useful for discovering how many neighbors of each node there are, and allows us to use a queue or a deque.

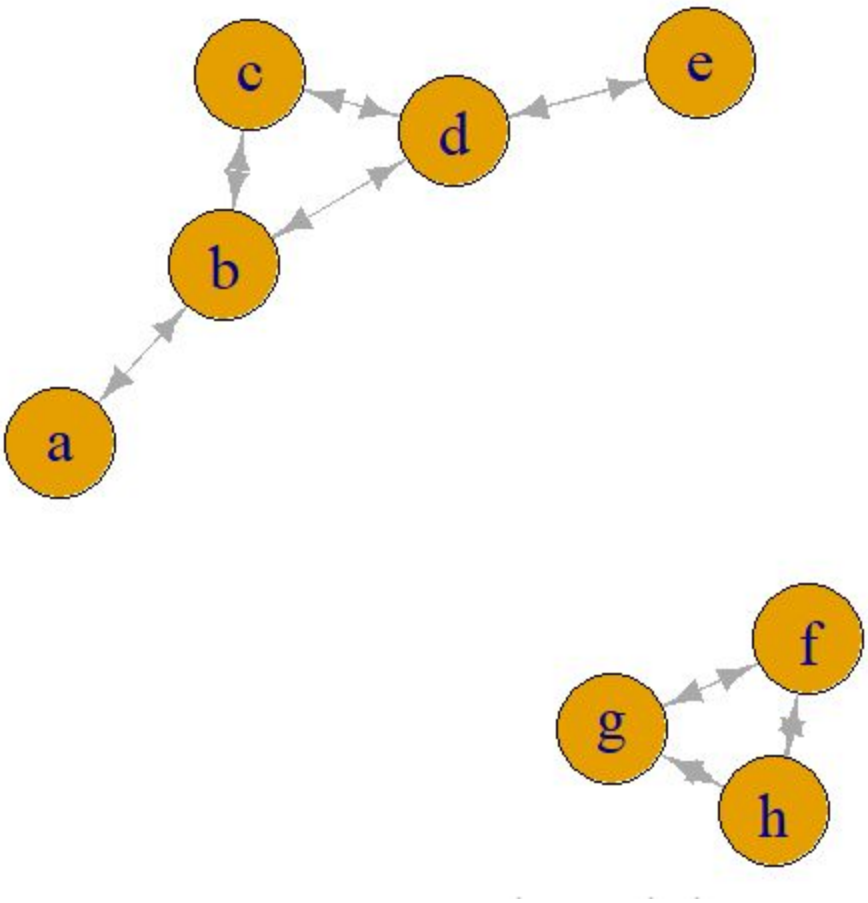
METHODS

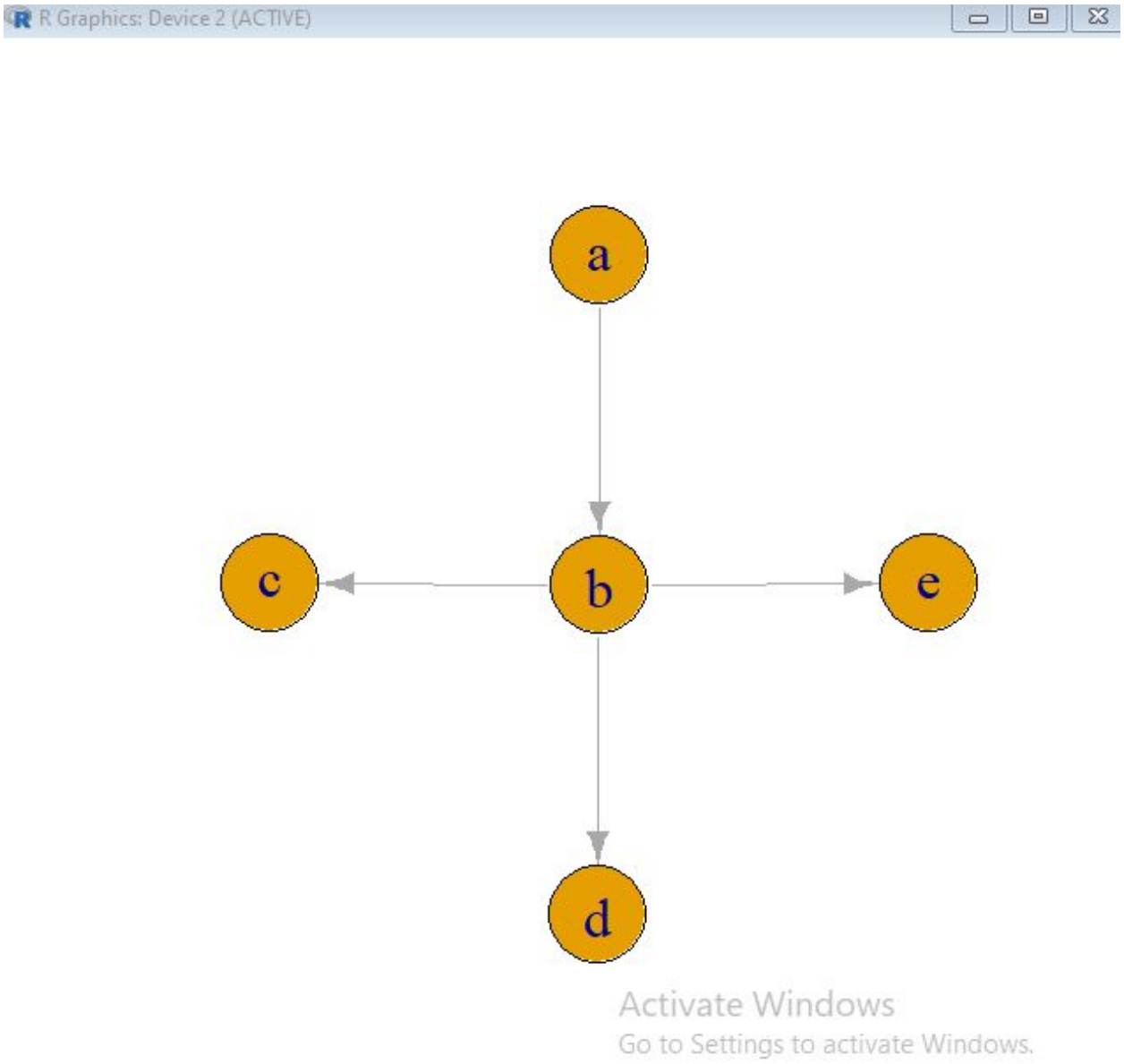
I implemented a BFS algorithm using pseudocode from the textbook, and a deque. I used a queue at first, and after some online research, decided that a deque may be faster, and more versatile, than a queue (apparently a deque is a container, but a queue is not; therefore, more functionality is achieved with a deque rather than a queue).

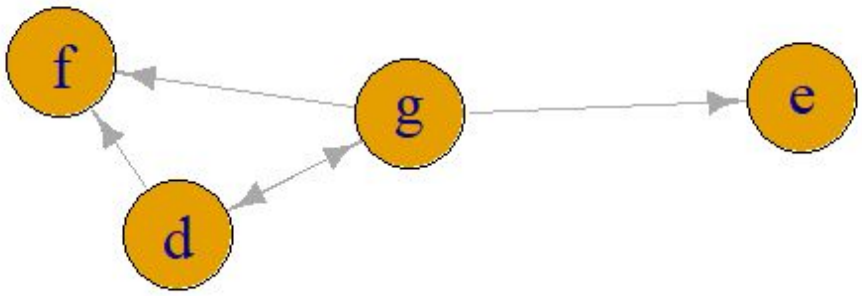
I tested using a test_BFS function, which tested all five of my test graphs (which varied from acyclic/cyclic, and undirected/directed) and compared the results to what I calculated on paper (and inserted into a vector for comparison).

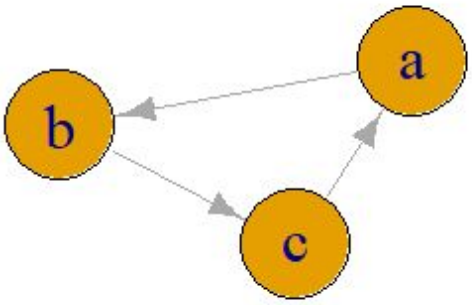
RESULTS

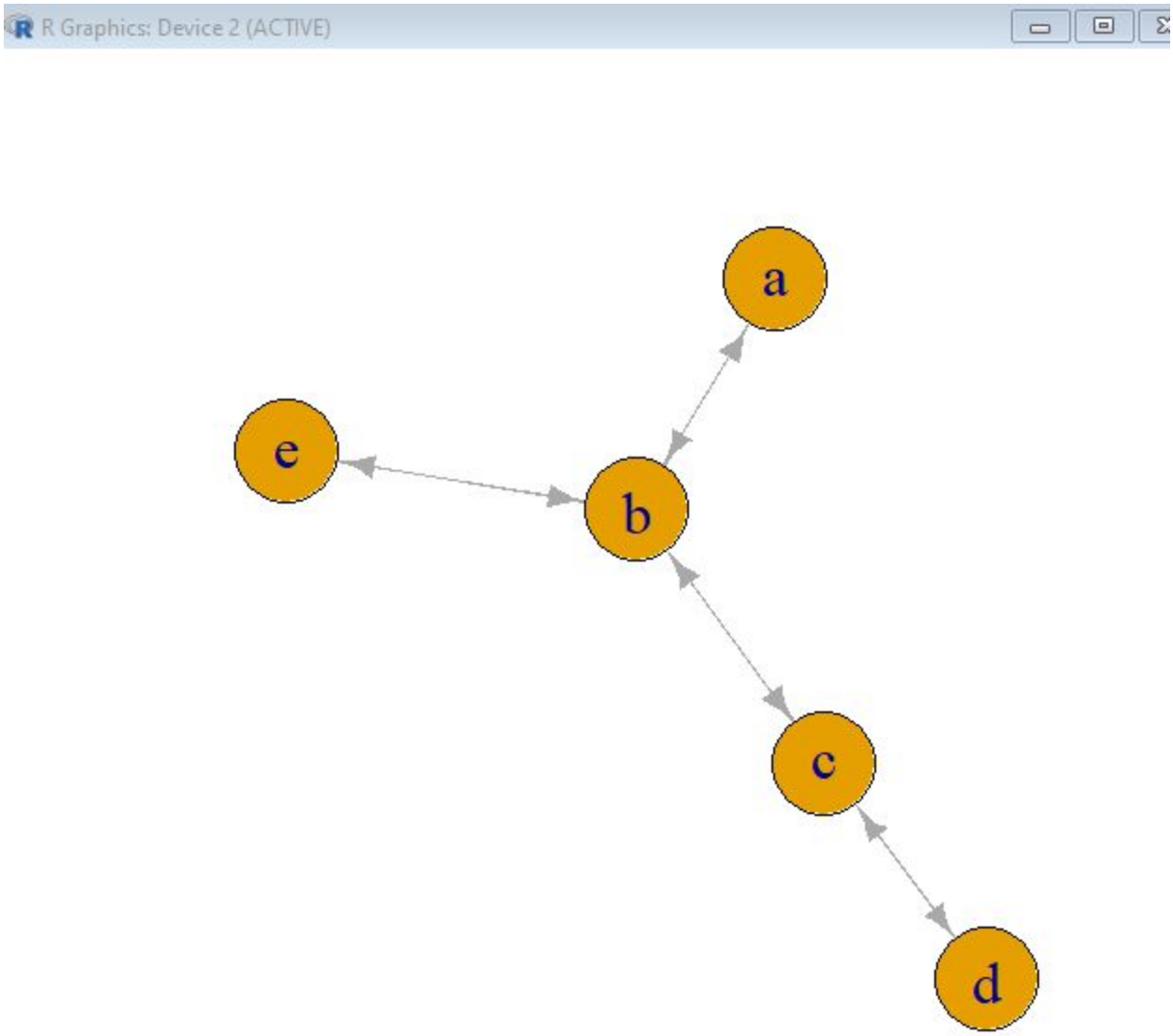
The following are my five test graphs, un_c_2, dir_ac_3, dir_c_2, un_ac_1, un_ac_2, respectively:

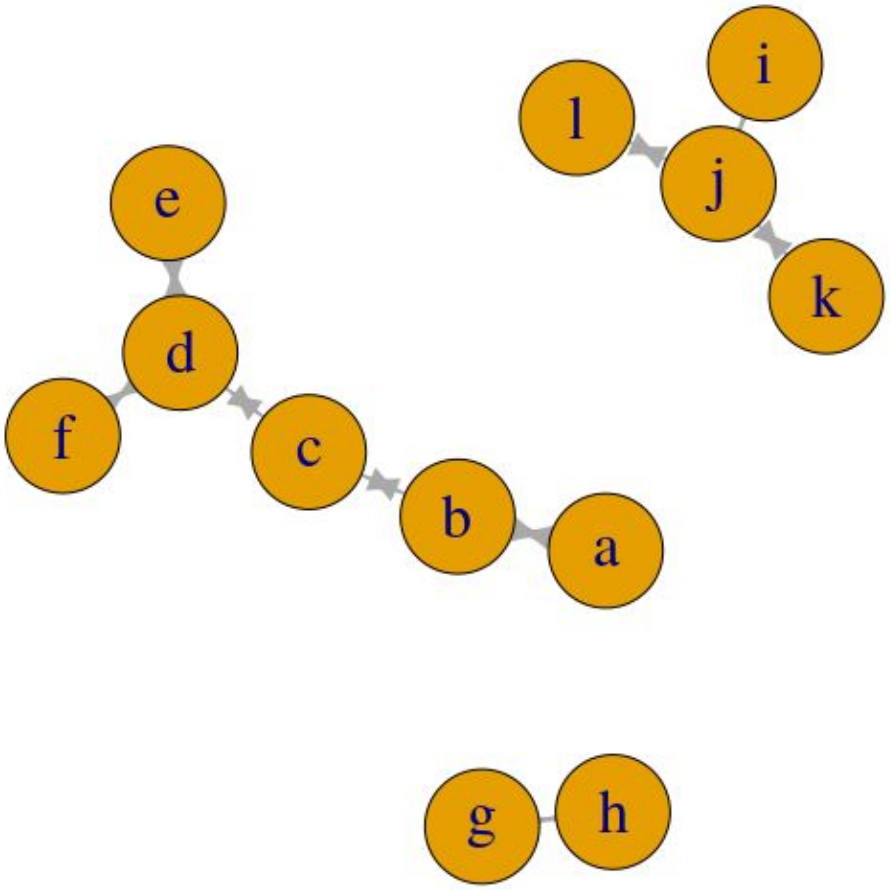

c
e
d
b
a
f
g
h


R Graphics: Device 2 (ACTIVE)
a
c
b
e
d
Activate Windows
Go to Settings to activate Windows.





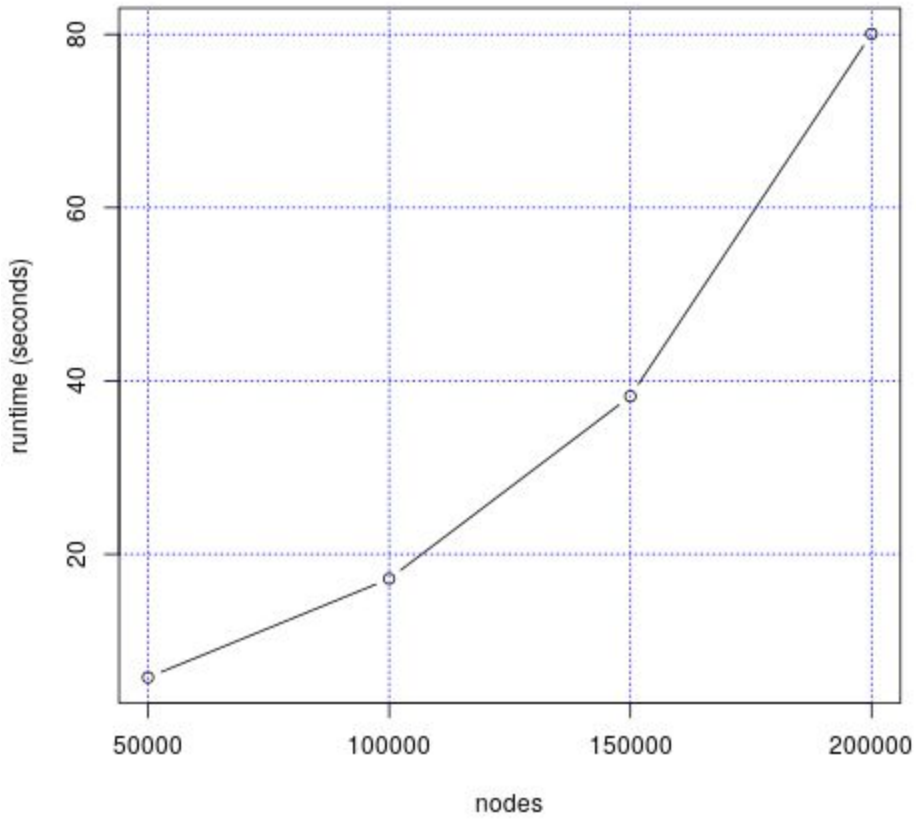

R Graphics: Device 2 (ACTIVE)
a
e
b
c
d
Activate Windows
Go to Settings to activate Windows.

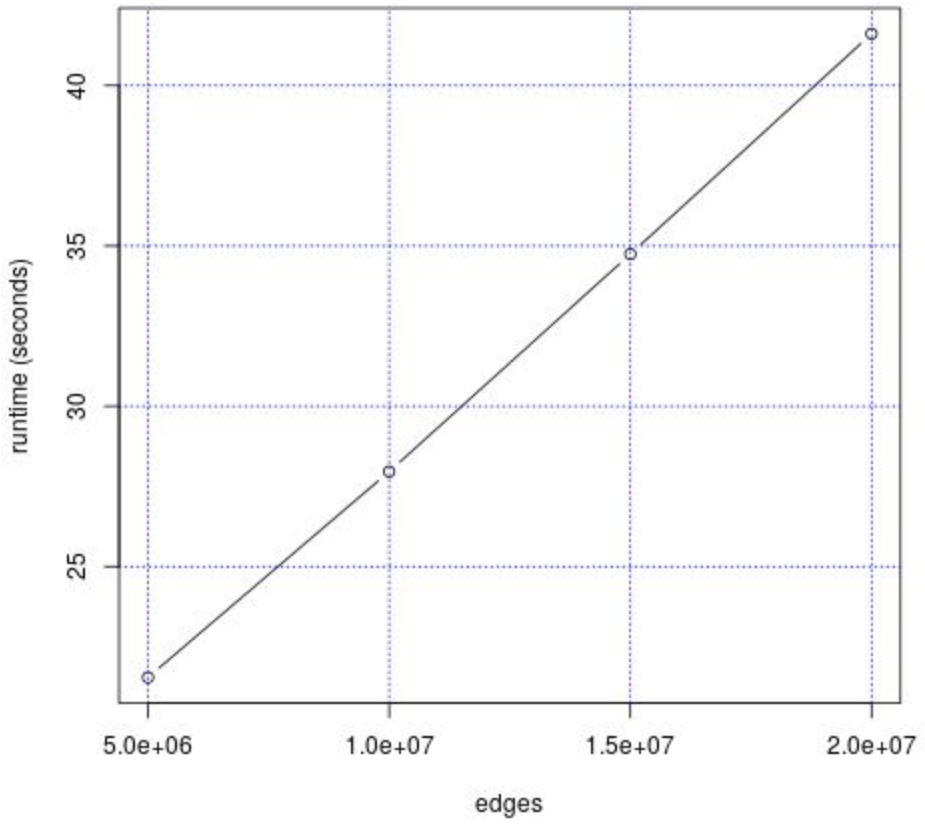


Each test case passed.

I generated 4 large graphs of nodes, each with 2,000,000 edges. The runtime on these graphs is illustrated in the first graph, with 'nodes' as the xlabel. I also generated 4 large graphs of edges, each with 100,000 nodes. The runtime on these graphs is illustrated in the second graph, with 'edges' as the xlabel. Each were generated using R code in 'main.cpp'.

The following are my results on extremely large numbers of nodes and edges:


runtime (seconds)
20
40
60
80
50000
100000
150000
200000
nodes



The function of nodes graph appears to be quadratic. Initially, I used the STL queue data structure to implement BFS in my program, however, the results looked quadratic, and I figured that it was probably caused by std::queue::push, as the documentation on cplusplus.com says that 'push_back' is used. From previous experience, push_back has needed to reallocate memory for every element before adding a new element.

Since cplusplus.com mentioned a deque had a constant time push_back function, I tried using a deque. However, it appears the problem may not have been using a queue, as the graph still looks quadratic after replacing a queue with a deque.

The function of edges appears to be linear. This makes sense, as iterating through edges in the adjacency list doesn't seem like it would be an O(n^2) operation.

DISCUSSION

The runtime of my BFS function is useful for graphs with large numbers of edges, but not as many nodes. I suppose that I could create my own data structure, like my own queue, to confirm whether or not the data structure of choice is causing runtime issues.

I think it might be something in my Graph class, but I will have to do some deep searching for it.

For a lot of situations, my BFS function could be used, without any problems due to runtime. But in a big data setting, O(n^2) is typically unacceptable.

CONCLUSIONS

There is an extremely high importance in finding algorithms that run quickly for large input. Large graphs are encountered fairly often in computer science, so it is important to read, analyze, and save them efficiently.

Utilizing proper data structures for maximizing efficiency for various tasks is very valuable. For example, using a queue instead of a vector has made it possible for BFS to be run on huge input.