NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ2001: Algorithms

AY20/21

Lab Group: SSP1

Lab 2 Report

Group Members:

LIM JIA EN (U1922802B)

NG CHI HUI (U1922243C)

KOO JIAN YANG (U1921728K)

RUSSELL LEUNG (U1920170K)

SUHANA GUPTA (U1923230B)

Introduction

In this report, our objective is to find a shortest path algorithm where the time complexity of the algorithm is independent of the ‘k’ value of hospitals that is found. Thereafter, returning the shortest ‘k’ hospitals. For the project, we had modified the Breadth-First-Search Algorithm to ensure originality and execution of the program.

Description of BFS Algorithm

Considering the graph is unweighted and undirected, we have shortlisted and implemented the BFS algorithm as it is able to find the shortest path from start node to the end node. In addition, we have also used the concepts of BFS to satisfy part a, b, c and d of the lab manual.

The implementation of the algorithm consists of 3 different data structures that were taught which consist of the LinkedList, HashMap and HashSet.

-LinkedList is the temporary storage of the next nodes to be checked in upcoming iterations.

-HashMap forms connections between the adjacent nodes back to the vertex that is connecting to the nodes.

-HashSet keeps a record of the nodes that have already been iterated, to prevent iterated nodes from being checked again. This saves the overall time complexity of the algorithm.

Our algorithm implements the above data structure, by allowing us to form and create a path by working backwards from the end node to the start node. Let ‘V’ be the path length, the currentNode which is paired with ConnectingNodes will be executed first. After the execution process the nodes will then return ‘V--’ nodes of the graph. This data is then in turn stored in a list of the HashSet. This process will then be repeated until the starting node is being traced. With the starting node being traced, the whole path will be reflected in the HashSet and output accordingly.

Implementation (a) and (b)

The implementation begins by instantiating the 3 different data structures we had listed above. If the Linkedlist is not empty, it will carry on running until the conditions turn false.

In the loop, the process of removing the first node at the first index of the list is done and then stored as the currentNode. Thereafter, the algorithm will check if the node is present in the list that stores the list of Hospitals. The loop will then continue executing to check for adjacent nodes if the currentNodes is not labelled and classified as a Hospital. Due to the graph size being huge, the presence of adjacent nodes are expected and if the adjacent nodes are not being visited it will be then stored into the LinkedList and the path will be outputted into the HashSet with the assistance of HashMap to form the connection of the graph.

Analysis for part (a) and (b)

Using the graph terminology and convention, let ‘V’ be the number of vertices and ‘E’ be the number of edges.

From the graph analysis as shown in Figure 1, the number of hospitals which are represented by 'k' are highlighted in blue and the source node highlighted in green.

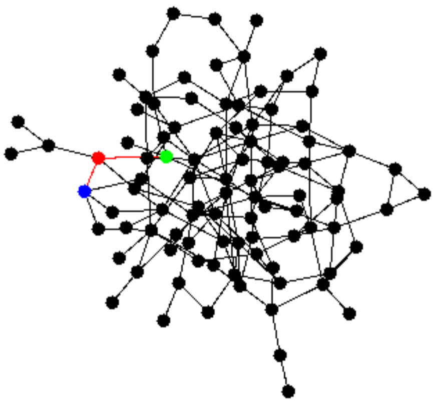
Figure 1: Path to nearest Hospital when k=1

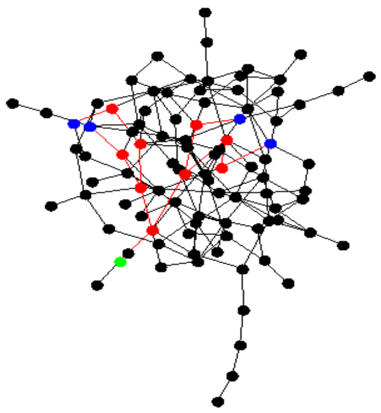
Figure 2: Path to nearest Hospital when k=4

From Figure 1 and 2, the algorithm searches the green dot, then expands to its adjacent nodes. This repeats until one node matches or is connected to one of the blue nodes, thus forming a path.

For a graph that is unweighted and undirected, the time complexity of the search algorithm using the BFS is O(V + 2E) with V being the vertices and E being the edges which can be simplified to O(V+E). This is the average case time complexity when the graph is sparsely connected, where the number of edges is less than vertices. The complexity derived is such that each vertex is only explored once, and each edge being explored twice since this is an undirected graph. In the road network dataset that is obtained from the Stanford website, the graph might consist of multiple nodes connecting to one another such that the graph is fully connected, meaning that it is a complete graph. Taking note of the vertices being possibly connected to multiple vertices, the time complexity of using the BFS algorithm is O((V+(V^2 - V)). This is because the number of edges for a fully connected graph is ½* |v|*|v-1|. Thus when substituted O(V+E) = O(V+½* |v|*|v-1|) = O(V^2). The presence of (V^2 - V) in the equation is because BFS must go through all nodes when the graph might consist of multiple nodes connecting to one another.

<u>Implementation of (c) and (d)</u>

To satisfy the implementation of these (c) and (d) parts in the lab manual we have made slight adjustments to the algorithm. Our algorithm works by making another copy of the list containing the hospital nodes. In the copy of the list, we drop the hospital nodes that have been found, making the hospital nodes to normal nodes. Thus, amounting to k - (1 * iteration index). This conversion of a meaningful node to a normal node can be done using what was taught in the lecture about the concepts of BFS using d, pi and s. In the lecture, the nodes are converted by passing the list of hospital nodes as a parameter. By removing the hospital, the node that was once a hospital will return a Boolean value of 'False'.

In this example, we would be providing a list of an array of 20 different value representing the vertices for hospital:

[57, 424, 357, 423, 138, 57, 35, 246, 467, 64, 4, 74, 322, 300, 355, 257, 168, 367, 189, 149]

The values reflected in the array list are the hospital nodes in the graph.

If the source node starts at node number 357, it would then be removed from the list and assumed to be a normal vertex in the graph. In addition, if vertex 357 is the first element in the LinkedList, it will be the current vertex for the next iteration.

Analysis of part (c) and (d)

The change in our algorithm that we had come up previously will reflect the following changes in the time complexity that is larger than equal to O(V + 2E) * k and smaller than equal to O(V^2) * k.

The time complexity of the algorithm will always be independent of the hospital present because the BFS algorithm will search through all the nodes level by level regardless if the hospital nodes exist or not. Our algorithm is also capable of finding top k hospitals, which satisfies (c) and (d).

Using Excel, our team set out to find the correlation between 'k' number of hospitals and the time taken to complete as shown in Figure 3.

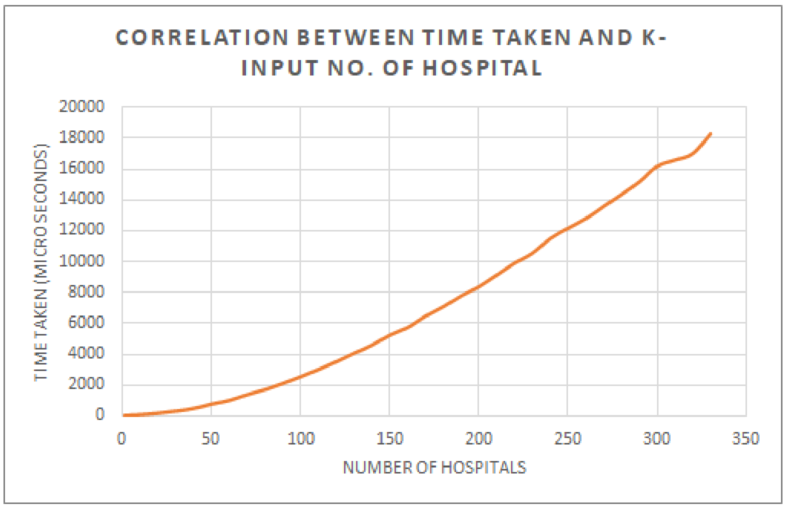

Figure 3: Correlation between Time Taken and K-input no. of Hospital

From Figure 3, we can see that there is a linear correlation. As the 'k' number of hospitals increases, the runtime for the algorithm to complete also increases linearly.

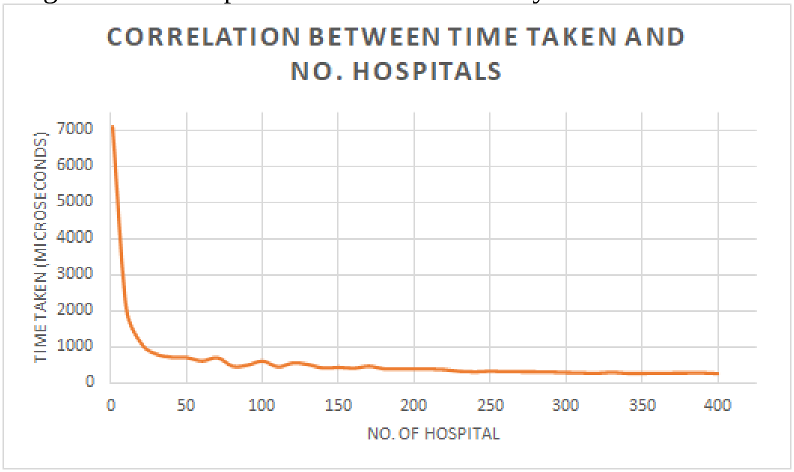

Figure 4: Correlation between Time Taken and No. of Hospital

In Figure 4, we notice that the time for the algorithm to complete was constant when the number of hospitals is more than 175. This shows the inverse relation between the runtime of the algorithm and the number of hospitals present.

Conclusion

A graph traversal is a unique process that requires the algorithm to visit, check, and/or update every single un-visited node in a tree-like structure. The BFS algorithm that we have suggested works on a similar principle. It is useful for analysing the nodes in a graph and constructing the shortest path of traversing through these.

BFS selects a single node and makes that as a source node in a graph and then visits all the nodes adjacent to the selected node.

We also consider that most road network graphs are not undirected and therefore this might be a flaw that should be accounted for if this algorithm is implemented.

Team Contribution:

Name:	Contribution:
Lim Jia En	Code Implementation and Report writing
Russell Leung	Code implementation
Ng Chi Hui	Code Implementation and Report Writing
Koo Jian Yang	Code Implementation
Suhana Gupta	Code Implementation and Report Writing

References:

R. Megha, "Graph Search: Breadth First Search - Lets Code Them Up!", Lets Code Them Up!. [Online]. Available: https://www.letscodethemup.com/graph-search-breadth-first-search/. [Accessed: 26- Oct- 2020].

"Breadth First Search or BFS for a Graph - GeeksforGeeks", GeeksforGeeks, 2020. [Online]. Available: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/. [Accessed: 25- Oct- 2020].

K. Moore and A. Gujrati, "Breadth-First Search (BFS) | Brilliant Math & Science Wiki", Brilliant.org. [Online]. Available: https://brilliant.org/wiki/breadth-first-search-bfs/. [Accessed: 27- Oct- 2020].

"Analysis of breadth-first search (article) | Khan Academy", Khan Academy. [Online]. Available: https://www.khanacademy.org/computing/computer-science/algorithms/breadth-first-search/a/analysis-of-breadth-first-search. [Accessed: 28- Oct- 2020].

P. Garg, "Breadth First Search Tutorials & Notes | Algorithms | HackerEarth", HackerEarth. [Online]. Available: https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/. [Accessed: 29- Oct- 2020].