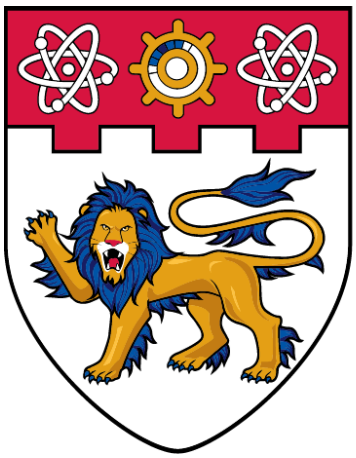


NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

CZ2001: Algorithms

Lab Project 2

Submitted By:

Jaheez Aneez Ahmed

Lim Jin Yang

Taneja Parthasarthi

Tan Hu Soon

Wu Jun Yan

A) INPUT DATA

Source: Road Network Data from https://snap.stanford.edu/data/index.html#road(.txt files)
Graph Library Used - `networkx`
Maximum Input File Size : roadNet-CA file (85.7MB) with the below device specifications.
Device Specifications (RAM : 12 GB, Processor : Intel Core I7, CPU : 64-Bit Architecture)

We would also like to state that throughout the report, we will take **n as the number of nodes in the graph, k as the number of hospitals to be searched and e as the number of edges in the graph.**

B) ALGORITHMS

1. INTRODUCTION TO BREADTH FIRST SEARCH:

BFS is a traversing algorithm where traversal starts from a selected node (source or starting node) and goes through the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). It then moves towards the next-level neighbour nodes.

2. INTRODUCTION TO MULTI SOURCE BREADTH FIRST SEARCH:

The Multi Source Breadth First Search (BFS) is a type of modification of BFS. In traditional BFS, the algorithm uses a user defined source node and will start by exploring all the neighbour nodes (nodes that are directly connected to the source node) before moving on to explore the next layer of nodes to find the nearest hospital node.

But for our algorithm Multi Source BFS is used. The implementation is done using a queue structure that follows the First-In-First-Out (FIFO) method. The hospital nodes will act as source nodes and will be put at queue first rather than a single node which is the case for traditional BFS. For example, if there are ‘h’ hospital nodes, then these ‘h’ hospitals the roots of ‘h’ different BFS searches that will start concurrently. After which, the sub - nodes which originated from different source nodes will be placed into the same queue in accordance with their depth level, which results in the elimination of the problem of nodes mistook another source node ‘h’ as its nearest k-neighbour after satisfying the k-neighbour condition.

3. IMPLEMENTATION: a) AND b)

We make use of Multi Source BFS to find the nearest hospital available for each of the Nodes. We add the hospital nodes into the BFS queue and carry out BFS one level at a time for each of the hospital nodes. We dequeue the first hospital node, traverse the graph to all adjacent nodes and enqueue all unvisited nodes. The shortest distance from the nodes to the hospital has been found for the unvisited nodes and we mark the nodes

as visited to prevent double counting of nearest hospitals. Once all adjacent nodes have been visited, we will dequeue the second hospital and carry out BFS. This is to ensure the distance obtained is always the shortest distance as the nodes reached by each hospital is the same at any point in time. We will iterate through the graph until all nodes have been visited and we have hence found the shortest path for all nodes in the graph.

4. IMPLEMENTATION: c) AND d)

This is an extension of a) and b) where the number of times a node is allowed to be visited is changed. We allow for every node in c) and d) to be visited k times as we want to find the shortest distance and path between the node and k number of hospitals. There will be 3 possible cases we can encounter while traversing the graph, the first one being that the node visited has already been visited by our root node. We will skip searching this node to prevent repeats as we are looking for k distinct distances and paths. The second case being that the node has already been visited k times. The node here has already found k nearest hospitals and we will mark the node as a dead end and skip it. The last case is that we encounter a node that has not been visited by our root node and has not been visited k times. We indicate that one of the k nearest hospitals is our root node and we will prepare to visit its children for the next iteration. The node will be marked as visited to ensure double counting does not happen. This process will carry on until all nodes have been visited by k number of hospitals.

C) TIME COMPLEXITY ANALYSIS

For parts ‘a’ and ‘b’, the best and worst case time complexity is **O(n+e)** as the hospital source node needs to traverse through all the nodes to obtain the shortest path of each node relative to the hospital source node location.

In an adjacency dictionary, every node is processed once while every edge is considered twice for an undirected graph. The time complexity will ignore the constant 2 from O(n + 2e), hence the asymptotic time complexity is **O(n + e)**.

For parts ‘c’ and ‘d’, the best and worst case time complexity is **O(k(n+e))** as the variable ‘k’ is included in the time complexity which essentially is the number of nearest hospitals that each node must find. As ‘k’ increases, each node must find and store ‘k’ number of hospitals and paths in hFound.

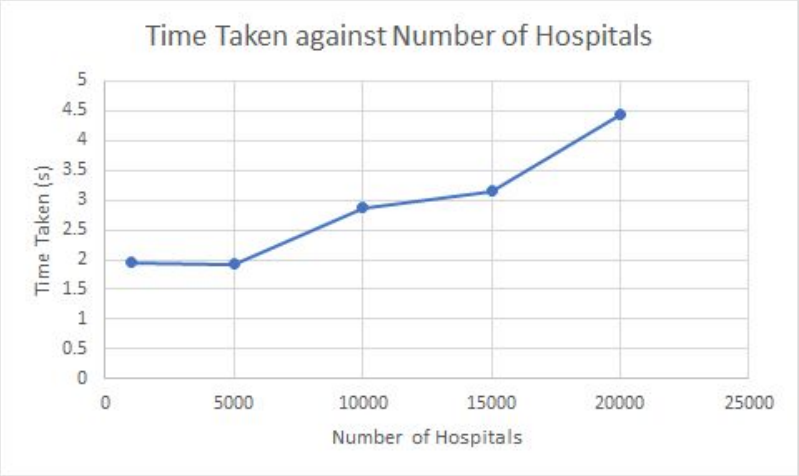
Therefore, for every nodes n and edges e, it will be explored k times. This results in the asymptotic time complexity of **O(k(n + e))**.

D) EMPIRICAL ANALYSIS

As stated in the beginning of the report, there are three main variables here which are number of hospital nodes (h), number of hospitals to be searched (k), and the file size (number of nodes and edges).

The variables will be varied one at a time while the rest of the variables remain constant. By analysing the data and time taken to run the algorithms, we can practically check the validity of theoretically derived time complexity.

VARYING *h*, THE NUMBER OF HOSPITALS:



Using a file with 100,000 nodes and keeping k as 1 and varying the number of hospitals from 1000 to 20000 at intervals of 5,000.

As observed from the graph, the time taken for the algorithm does not vary much with the varying number of hospitals. It can be concluded that the time taken to find a path from every node to the hospital is independent of the number of hospital nodes.

VARYING *k*, THE NUMBER OF HOSPITALS TO BE SEARCHED:

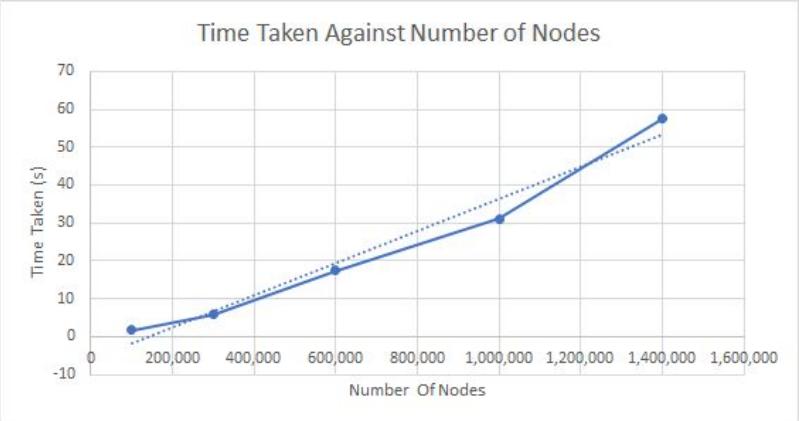
Using a file with 100,000 nodes, we used 5,000 nodes as hospitals and varied k-input from 1 to 1,000 in intervals of 50. The time v/s k-input plot is shown below:

In order to determine the nearest k hospitals and the distances for those hospitals from the node, the algorithm will have to find k different paths from the node to the k different

hospitals. The number of paths to search will increase, resulting in an increase in the time taken to search for all the possible k paths.

And as observed from the graph, the time taken to run is linearly dependent on k as proved earlier by the theoretical deduction of time complexity i.e. **O(k(n + e)).**

VARYING *n+e*, THE FILE (GRAPH) SIZE:



We used nodes which varies from 100,000 to 1.4million which are randomly generated and fixed 5000 nodes as hospitals in each. K-input is also set to 1 for each file.

It can be deduced from the data that the time taken for the algorithm increases linearly as the total number of nodes and edges increase.

Based on the algorithm, the number of nodes in the queue will increase as the file size increases. This will result in *n* number of nodes to be searched. Hence, resulting in a linear increase in the time taken to search as the number of nodes and edges increases.

This conclusion checks out with the theoretical deduction of time complexity which states the linear dependence of time taken on the sum of the number of nodes and edges as **O(k(n + e)).**

E) REFERENCES:

1. https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/#:~:text=BFS%20is%20a%20traversing%20algorithm,the%20next%2Dlevel%20neighbour%20nodes. (BFS)
2. https://www.geeksforgeeks.org/multi-source-shortest-path-inunweighted-graph/ (Multi Source BFS)

F) CONTRIBUTION STATEMENT:

"We all declare that we have contributed equally towards researching, implementing, testing, reporting and presenting the algorithms. We have also referenced all the work we

took inspiration from, to the best of our abilities.”