****

**CZ2001 Algorithms**

**Lab Project 2**

**Graph Algorithms**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Index Group: SS4

Group: 2

Members: Selvira Junita Melia (U1921926L)

Glenda Hong Zixuan (U1921784A)

Chia Jia Tian (U1921632K)

Ernest Ang Cheng Han (U1921310H)

Tan Wee Li (U1922935K)

[**Introduction**](#_yd5qx44efqkb) **2**

[**Correctness of Algorithm**](#_kuv5zuw7qoyq) **2**

[**Pseudocode**](#_vjkksrj5jkqi) **2**

[**Time Complexity: (a) & (b)**](#_noo6t55ju53q) **3**

[**Time Complexity: (c) & (d)**](#_7w83cvp4ltws) **3**

[**Space Complexity**](#_z76gcl7q5oz0) **4**

[**Empirical Analysis**](#_rd1v2wjzamv7) **4**

[**References**](#_26c0sfghwl9z) **5**

# 

# **Introduction**

For our project, we decided to create algorithms based on Breadth-First Search Algorithm to tackle the problem. It is an algorithm that is used commonly for data structures represented by graphs.

Our algorithm starts the search at a certain starting node (sNode) in the graph (G) and explores every single neighbouring node of the same depth to find any hospital nodes before iterating to the next depth level.

# Correctness of Algorithm

Base Case: Shortest Distance = 1

If the shortest distance = 1, the nearest hospital must be 1 edge away from the source node. Hence, the nearest hospital must be a neighbour of the source node. Our algorithm will iterate through the neighbouring nodes of the source node and stop once it hits a hospital. Thus, our algorithm will be able to find the shortest path and the nearest hospital for shortest distance = 1.

Inductive Step:

Assume that our algorithm will be able to find the shortest path and the nearest hospital for shortest distance = d, where d is some positive integer. For shortest distance = d, the nearest hospital must be d edges away from the source node.

If the shortest distance = (d+1), the nearest hospital must be (d+1) edges away from the source node. Hence, the nearest hospital must be a neighbour of a node that is d edges away from the source node. From the assumption above, our algorithm is able to pass through all nodes that are d edges away from the source node. Next, our algorithm will iterate through the neighbouring nodes of all these nodes and stop once it hits a hospital. Thus, our algorithm would be able to find the shortest path and the nearest hospital for shortest distance = (d+1). Hence, by mathematical induction, our algorithm is able to find the shortest path and nearest hospital for all values of the shortest distance.

# Pseudocode

|  |
| --- |
| **Breadth-First Search** |
| Program **INT** BFS(Graph, SourceNode)  **LET** q **BE** linked list (queue)  **FOR** int i = 0, 1, 2, 3, … (totalNodes) **DO**  traversed[i] = FALSE  distance[i] = **INTEGER.MAX.VALUE**  path[i] = -1  traversed[SourceNode] = TRUE  distance[SourceNode] = 0  q.add(SourceNode)  **WHILE** q is not empty **DO**  v = q.remove()  **FOR** i in linked list of node v **DO**  **IF** not traversed  SET traversed as TRUE  UPDATE distance  UPDATE path  ADD i to q  ***IF*** *i is a hospital node*  *Remove hospital mark on i (in Part C & Part D)*  **RETURN** i  **RETURN** -1 |
|

|  |
| --- |
| **Print Results** |
| Program **STRING** (sNode,dNode,tV,LinkedList)  **IF** sNode not connected to hospital node  **PRINT** “Source and destination not connected”  **ELSE**  Initialize empty LinkedList nodePath  nodePath.add(dNode)  Add subsequent nodes until sNode to nodePath  **PRINT** Shortest path length and distance from sNode to hospital node  **OUTPUT** into file using FileIO |
|

# 

# Time Complexity: (a) & (b)

**n → Number of nodes, m → Number of edges**

For our time complexity analysis, we are only considering the time taken for 1 source node to find the nearest hospital node for **(a) & (b)** or finding multiple nearest hospital nodes for **(c) & (d).**

|  |  |  |
| --- | --- | --- |
| **O(n+m)** | | |
| The algorithm will iterate through breadth-first search and may traverse the edges any number of times between 1 to ***m*** before finding a hospital node. | | |
|
|
| Best Case: O(1+1) = **O(1)**  Source node has a hospital node only 1 edge away. | Average Case: **O(n+m)**  Source node passes through ***n/2*** number of nodes and traverses ***m/2*** number of edges to reach a hospital node. Hence, time complexity is O(n/2 + m/2) = **O(n+m)**. | Worst Case: **O(n+m)**  Each edge is processed once in the while loop for a total cost of m.  Each node is queued and dequeued once, for a total cost of n.  Hence, time complexity is **O(n+m)** |

# 

# 

# Time Complexity: (c) & (d)

|  |  |  |
| --- | --- | --- |
| **O(k(n+m+1))** | | |
| After breadth-first search is run each time, demark the current nearest hospital. Hence, the next time breadth-first search is run, the next nearest hospital will be identified. The algorithm will iterate through the breadth-first search ***k*** number of times in total to find the top-k nearest hospitals. For part C, k = 2. | | |
|
|
| Best Case: **O(k)**  Source node has k hospital nodes only 1 edge away. | Average Case: **O(k(n+m+1))**  Source node in the graph has to traverse an ***m/2*** number of edges to find the each hospital node. ***n/2*** nodes are also traversed when finding each hospital node. In each iteration, an operation is performed to remove the current nearest hospital node. Thus, total time complexity is O(k\*(n/2+m/2) = **O(k(n+m))**. | Worst Case: **O(k(n+m+1))**  Source node in the graph traverses the maximum number of nodes ***n*** and number of edges ***m*** before reaching each hospital node.  Each node is queued and dequeued once, for a total cost of n. The remove operation is performed similarly as the average case. The search is carried out k times, hence **O(k(n+m))** |

# 

# Space Complexity

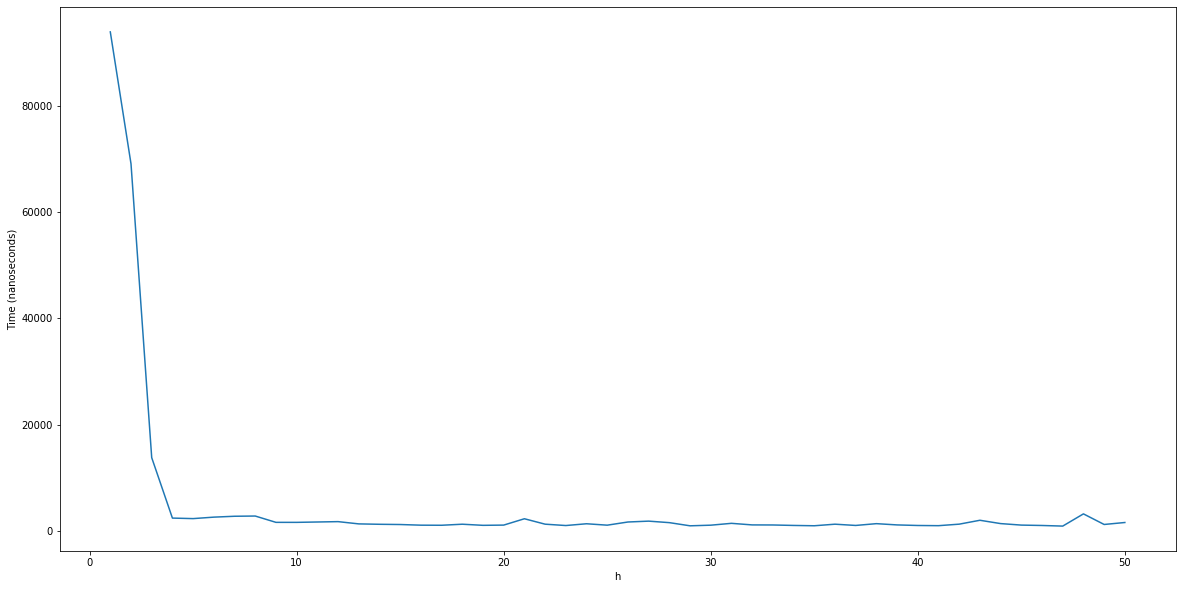
As the number of nodes are known, the algorithm will require O(n) space, ***n*** being the number of nodes. O(m \* k) = O(mk) space is required for each node depending on the number of hospitals (k) to be searched. Once the result is printed and exported into the output file, the space used is reset and can be reused for subsequent nodes.

# Empirical Analysis

To test the effect of h on the time complexity, the algorithms are run on the Small\_Input\_Graph.txt with hospital nodes stored in Test\_Hospital\_List.txt. The time taken to run the algorithm is measured for values of h from 1 to 50. The nearest hospital is always included in every test. For each value of h, the algorithm is run 100 times and the average time taken is written to a corresponding csv file.

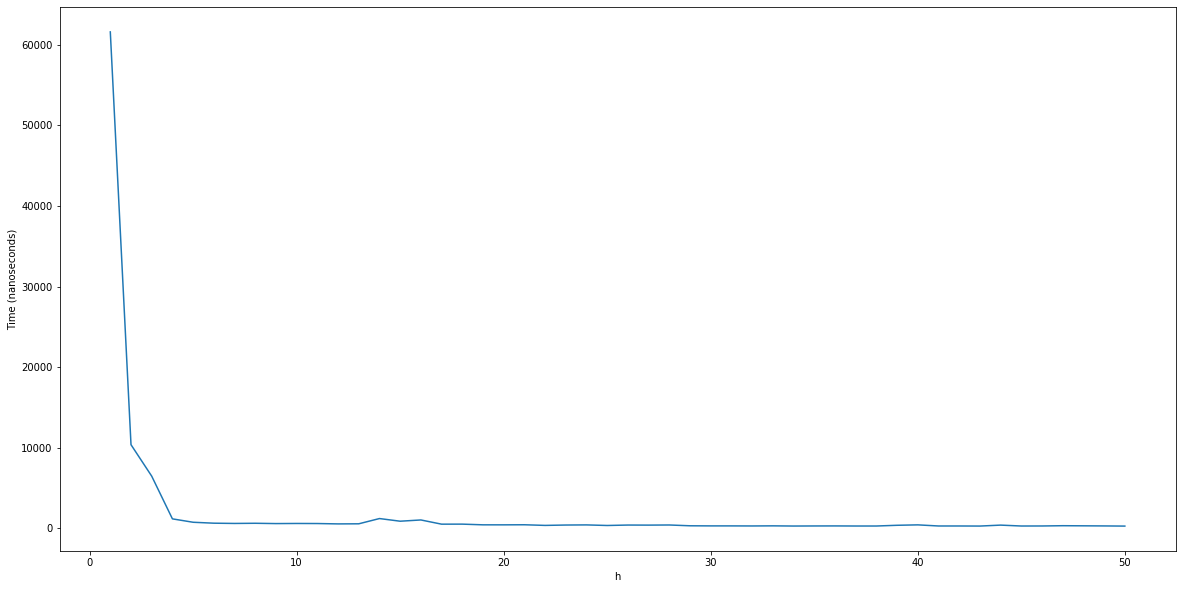
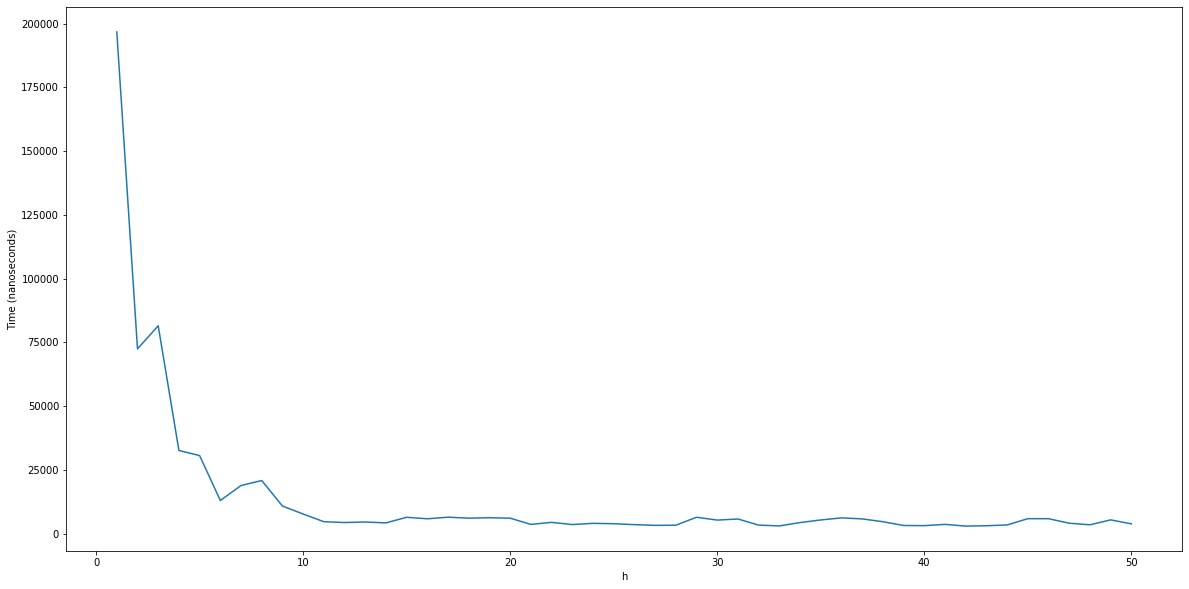
To test the effects of k on the time complexity, the part D algorithm is run on the Small\_Input\_Graph.txt with 50 hospital nodes stored in Test\_Hospital\_List.txt. The time taken to run the algorithm is measured for values of k from 1 to 50. For each value of k, the algorithm is run 100 times and the average time taken is written to dataDk.csv.

Effects of h on runtime for Part A & B

****

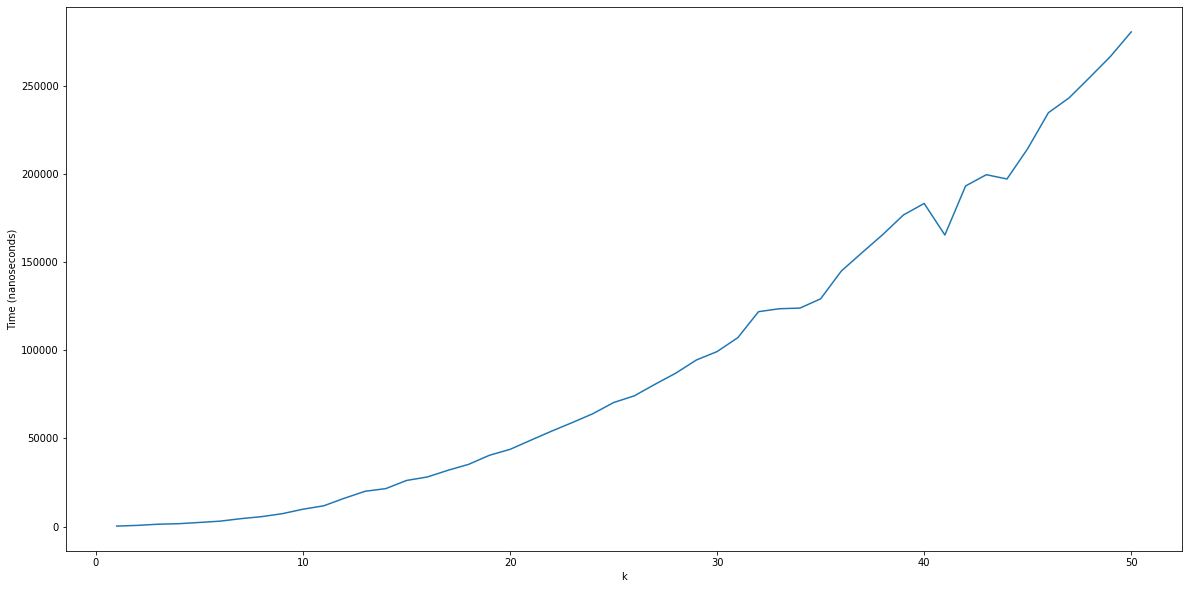
In general, the time remains about the same as h increases. Thus, the time complexity for Part A & B is independent of h.

Effects of h on runtime for Part C & D

****

For this empirical analysis, k was 2 for Part C and k was 20 for Part D. In general, the time remains about the same as h increases for both algorithms. Thus, the time complexity for Part C and D is independent of h.

Effects of k on runtime for Part D

****

The time increases proportionally as k increases. Thus, the time complexity for Part D is linearly dependent on k.

Discrepancy between theoretical analysis and empirical findings for h

When h is small, the time in our empirical findings does not follow our theoretical time complexity, instead the time decreases as h increases. This is because our breadthFirstSearch algorithm stops exploring paths when it hits a hospital node. Hence, at smaller values of h, the algorithm explores more paths since it hits less hospitals. This means that the discrepancy in time is due to the location of the hospitals relative to the source node, instead of the number of hospitals.

Conclusion

In conclusion, the empirical findings are consistent with our theoretical analysis.

# 

# References

1. Guru99 (7 October 2020). *Breadth First Search (BFS) Algorithm with EXAMPLE.* Retrieved 25 October 2020, from <https://www.guru99.com/breadth-first-search-bfs-graph-example.html>
2. Bo Waggoner (Fall 2020). Colorado CSCI 5454: Algorithms | Lecture 3: Shortest Paths. Retrieved 25 October 2020, from <http://www.bowaggoner.com/courses/2020/csci5454/notes/lect03-shortest-paths.pdf>

**Statement of Contribution**

|  |  |
| --- | --- |
| Name | Contribution |
| Ernest Ang Cheng Han | * Algorithm for Part A & B * Contributed to the lab report * Contributed to the slides * Presenter |
| Glenda Hong Zixuan | * Empirical Analysis * Contributed to the lab report * Contributed to the slides |
| Selvira Junita Melia | * Algorithm for Part D * Pseudocode * Contributed to the lab report * Contributed to the slides * Presenter |
| Chia Jia Tian | * Algorithm for Part C * Java Program Interface * Pseudocode * Contributed to the lab report * Contributed to the slides |
| Tan Wee Li | * Time complexity analysis * Pseudocode * Contributed to the lab report * Contributed to the slides |