CZ2001 Algorithms
AY20/21 Semester 1

Project 2 Written Report
Group 1

Member 1	Li Haocheng (U1921700J)
Member 2	Joey Lim Soo Yee (U1921745L)
Member 3	Josephine Agatha Hemingway (U1920309C)
Member 4	Kenn Lim Zheng Jie (U1921807J)
Member 5	Shannon Kate Wong Carlynn (U1921906D)
Member 6	Leong Hao Zhi (U1920469K)

Table of Contents

1. Introduction
2. Algorithms
    - 2.1 Main
    - 2.2 Algorithm A
        - 2.2.1 Analysis
    - 2.3 Algorithm C
        - 2.3.1 Analysis
3. Empirical Analysis
4. Conclusion
5. References

1. Introduction

In this written report, we explain and analyze the programme we designed to compute the shortest path from each node to the nearest hospital and find the k-nearest hospitals from the source node. consisting of two main search algorithms.

2. Algorithm

2.1 Main

To construct a road network graph, node-to-node information is first read from the real road text file (or *randomNodes.txt* file if a random graph is used). Node information will then be used to create and add edges into an adjacent list.
The **HospitalUserInput class** retrieves user input values for the total number of hospitals on the map as well as the respective hospitals NodeID and stores this information in a text file (*hospital.txt*). An ArrayList, *nodeIdAr* is then created in the main driver, reading in values from *hospital.txt* and storing the corresponding hospital node IDs in *nodeIdAr.*

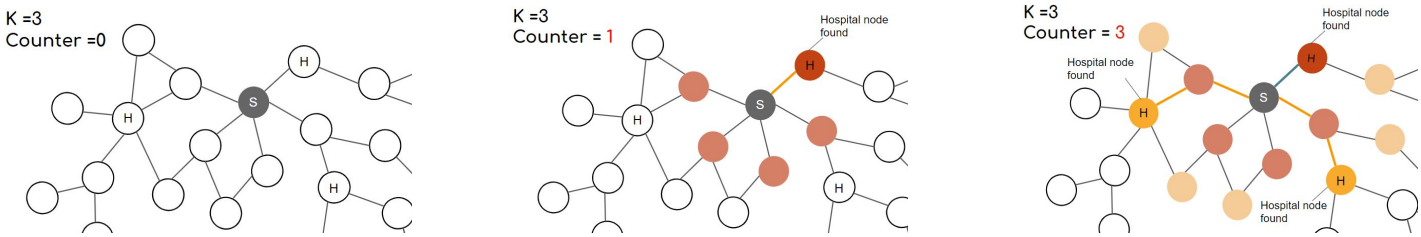
Hospital Class is used to store the attributes of the hospital: *nodeId*, *distance* and the *shortest path* to the nearest hospital from a given source. After obtaining the information for the nearest hospital from a single source by applying the modified breadth-first search (BFS), a for-loop is used to iterate for every input source (each node in G). The results are stored in the output file *(Output_A.txt or Output_B.txt)* which displays the source, nearest hospital, distance and its path.

2.2 Algorithm A

The conventional BFS algorithm explores all of the neighbour nodes at present depth prior to moving on to nodes at the next depth level and searches for the shortest path to a single hospital node. It then iterates through a loop, terminating only when all hospital nodes are explored to compute the shortest distances for all the hospital nodes and lastly compare and sort these results to obtain the *k*-nearest hospitals.

To improve efficiency and meet the requirement of part (d), we have designed algorithm A by modifying the BFS algorithm such that it only runs a single BFS algorithm and terminates when user-input *k* nearest hospitals to the source node have been found via a counter.

Our algorithm explores nodes which are closest to the source first. It first checks the first level of nodes that is connected to the source and adds them to the queue. Nodes of the next level that are found in the queue are de-queued and explored. When the node explored is identified as a hospital, that hospital's node ID, distance and path of the source to that hospital is stored as a ***Hospital object*** and the counter value is incremented by 1. This continues until the counter reaches *k* such that the *k* nearest hospitals are found for each source, storing each Hospital object into an array or when all nodes have been queued and explored.



As seen in the diagrams above, when user input *k-value* is 3, the algorithm will start traversing from *source S* to explore neighbouring nodes layerwise. In this case, when a node it explores has been identified as a hospital, the counter variable is increased by 1 to keep count on the number of hospitals found. This process continues and terminates when the counter is 3, showing that we have successfully identified the top 3 nearest hospitals from *source S.*

The modified BFS algorithm is then iterated in a loop, for each node of the graph as the source node, creating an instance of the Hospital array for each source node, which is then written into the output file.

2.2.1 Analysis of Algorithm A

Time complexity

Let *k* denote the number of nearest hospitals to find, *h* the total number of hospitals, *n* the number of map nodes, and *E* the total number of edges.

Best Case	Average Case	Worst Case
This occurs when the first *k* nodes searched are all hospitals, and the source node has these *k* nodes as its only nodes. The number of nodes visited and edges comparisons are both *k*. This simply gives a time complexity for each source node of *O(k+k)* $\approx$ ***O(k)*** For *n* source nodes: C = O(k) x n **= O(kn)**	Assuming that each node in the entire graph has an equal chance of being a hospital, and *h* >> *k* such that the probability of finding a hospital node is unaffected by any previously found hospital node. The probability of any node being a hospital node would be $\frac{h}{n}$. The total number of edges visited to find the *kth* hospital would follow a negative binomial distribution with parameters: (*k*, $\frac{h}{n}$). The mean is $\frac{k(n-h)}{h}$. Assuming each node has $\frac{E}{n}$ number of edges, the number of nodes for queue and deque is approximately $\frac{k(n-h)}{h}$ divided by $\frac{E}{n}$. adding comparisons made for each vertex, we have average time complexity for each source node**:** $c \times k(\frac{n-h}{h})(1+\frac{n}{E}) \approx O(\frac{n}{h} \times \frac{n+E}{E})$ = ***O(*** $\frac{kn^2}{h \times E}$ ***)*** For *n* source nodes: C = *O(* $\frac{kn^2}{h \times E}$ *)* x n **= *O(*** $\frac{kn^3}{h \times E}$ ***)***	This occurs when *h* is less than *k*, or when the last node searched is the *kth* hospital found, similar to a conventional BFS. Thus the entire graph is traversed, giving a time complexity for each source node of ***O(n+E)*** For *n* source nodes: C = O(n+E)) x n **= *O(*** n^2 **+ *nE*)**

Space Complexity

The space complexity here assumes the worst case for time complexity, where the entire graph is traversed. The space complexity is dependent on two parts: space for storing the graph itself and the space occupied by the queue storing the nodes that have been visited.

For storing the graph itself, we used an adjacency list which has space complexity O(n+E). The space complexity for storing the queue in the worst case is O(n), but we need to do this for n times to get the result for all sources. Assuming that the variable is not cleared after each run, this results in a space complexity of O(n^2). The total space complexity is, therefore, **O(** n^2 **+ E).**

2.3 Algorithm B

To meet the requirement of the part (b) where the search algorithm is independent of the number of hospitals h, we have decided to use the idea of the Floyd Warshall algorithm to implement a complete version of BFS algorithm where the shortest distance between all possible source-destination pairs are first computed and stored.

This implementation of running the BFS for all possible source-destination node pairs allows the algorithm to run without even specifying its source and destination nodes, (i.e. user input hospital nodes need not be initialized for the algorithm to run). Finally, after the user has specified the number of hospitals and hospital node IDs, we simply refer to the stored ArrayList of all the possible pairs, isolate the *Node Objects(created by the Node Class)* with the destination as the user input hospital nodes and store them into a new ArrayList, *allNodeHosAr* before sorting from the shortest distance. Lastly, we then isolate the top K objects in the list and add them into *OutputArray* We then run a loop to iterate this for all possible source node values to obtain a list of K*n items which displays the top K nearest hospitals for all source nodes.

2.3.1 Analysis of Algorithm B

Time complexity
Let *n be the* number of nodes in the map and *E* the number of edges.

In Algorithm B, the algorithm runs independently of the number of hospitals *h*, thus *h* is not included in its time complexity.

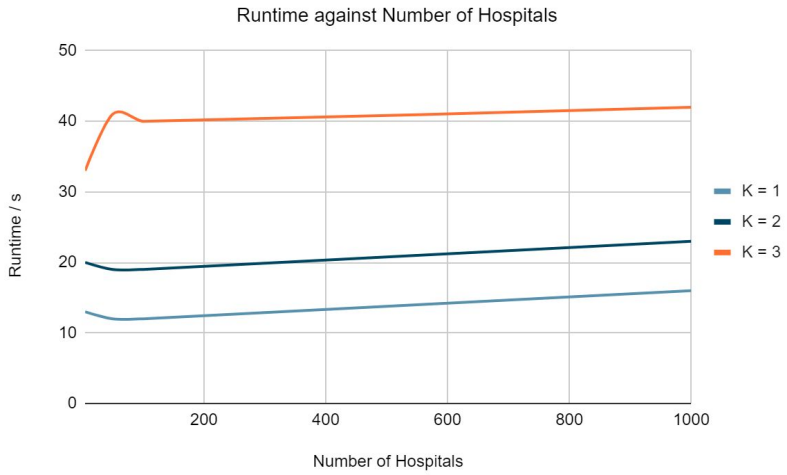
Each instance of BFS for 1 source node, all values are compared and objects are stored. Hence it runs with a time complexity of *O(n)*. We iterate this for all V source nodes to obtain the complete list of all source-distance pairs. Hence it runs with complexity *O(n)*.

For all cases, the time complexity is identical as the algorithm takes into account all possible permutations of source-destination pairs for obtaining the complete list and traverses down the entire list to isolate the required node objects. Hence, each iteration of the algorithm has a time complexity of *O(n+E)* to obtain all node objects for a single source and a total of *O(n(n+E)*) to obtain the complete list. For isolating the K nearest hospital nodes for all possible sources from the complete list, it would require another *O(*n^2*)*. Hence, total time complexity would be ***O(n(n+E) +*** n^2 ***)*** **= *O(*** n^2 **+ *nE*)**

Space Complexity
Space complexity for Algorithm B and Algorithm A is different only in storing the distance from each node to each other node. In our codes, we used an Arraylist in Arraylist(i.e. A matrix), which takes an additional O(n^2)space. The overall complexity is still **O(n^2+E).**

3. Empirical Analysis

Oregon Dataset: Nodes: 10670, Edges: 22003

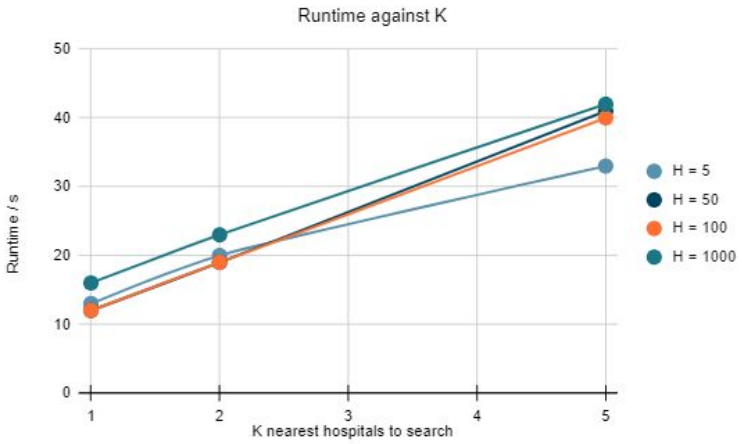


Effect of H on runtime

Conducting the empirical analysis, we can see that as the value of H increases, for all constant values of K, the runtime is relatively constant. This indicates that the time complexity is not dependent on H which contradicts our analysis.

However, when we consider the fact that H<<<n^3 and H<<<E, implies that the impact of H on time complexity is insignificant. Therefore

it is safe to assume that for most real application scenarios of this algorithm in the context where the number of hospitals is much less than the number of road networks, the time complexity is independent of H.



Effect of K on runtime

As observed, when the value of K increases, our time complexity increases almost linearly which aligns with the analysis of the time complexity. It is to be noted that variations in the actual run time is largely due to the actual location of the source node and the probability that it is located near K number of hospital nodes.

Additionally, assumptions made in the analysis of the average-case complexity may not hold for a real road network. Although it is safe to assume that the time complexity of algorithm A is independent of H due to its large value difference in real applications, for the purpose of evaluation where H can take on any values up to N, algorithm B was still designed such that the time complexity is independent of H.

4. Conclusion

Comparing our two algorithms, they are best used in different scenarios due to the difference in running time complexities. Algorithm A is more universal and more suitable to calculate the nearest hospital for different datasets, whereas algorithm B is most effective when calculating the shortest distance for changing start and destination nodes for the same dataset.

For example, algorithm A is suited for determining the nearest hospital for multiple node graphs like for the maps of Singapore, Malaysia and Indonesia.

On the other hand, algorithm B is suited for use on a single country's map like Singapore. After all the possible road connections are established, the shortest distance from any one location to another can be obtained most efficiently by simply searching for the source destination pair in the connections list.

5. References

Algorithm A
https://www.geeksforgeeks.org/shortest-path-unweighted-graph/
https://mkyong.com/java/java-check-if-array-contains-a-certain-value/

Algorithm B
https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/
https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

Member's Contributions

Group Member	Contributions
Li Haocheng	- Algorithm A - Algorithm B - Written Report
Joey Lim Soo Yee	- Graph Creation - Main - Algorithm B - Presentation & Slides
Josephine Agatha Hemingway	- Graph Creation - Main - Algorithm B - Presentation & Slides
Kenn Lim Zheng Jie	- Algorithm A - Algorithm B - Written Report
Shannon Kate Wong Carlynn	- Graph Creation - Main - Algorithm B - Presentation & Slides
Leong Hao Zhi	- Algorithm A - Algorithm B - Written Report