Nanyang Technological University

CZ2001 Algorithms Lab 2 Report

Lab SSP7 Group 7

Names
SIOW KEE TAT, KEITH (U1922767C)
STEFANIE LOW JING TING (U1920257C)
TING NAI XIANG, MATTHEW (U1922331J)
YEOW WEI LIANG, BRANDON (U1920258E)
TU XIANAN (U1921520L)

Introduction

In this project, we would like to compute the path from any point in a graph to its k-nearest hospital, where k is defined by the user. We used a floodfill inspired breadth-first-search algorithm to solve (a) & (b) and modified it to solve (c) & (d). We used the California road network dataset with 1,965,206 nodes and 5,533,214 edges and a randomly generated one.

Algorithms Design & Analysis

Requirement (a) & (b): Graph in Adjacency List Representation

```
1:   function bfs(graph, hospitals_list)
2:   begin
3:   create dictionary to path_nearest
4:   create visited_dictionary ← False for node in graph
5:   queue ← emptylist
6:   for each hospital in hospitals_list
7:      queue append (hospital)
8:      mark hospital node as True in visited_dictionary
9:      store a list of hospital in path_nearest dictionary[hospital]
10:   while (queue is not empty)
11:      path ← dequeue first list in queue
12:      curr_node ← first item in path
13:      for neighbour in graph[curr_node]
14:         if neighbour is False in visited dictionary
15:            create a new_path list from path
16:            add neighbour to start of new_path
17:            enqueue new path to end of queue
18:            mark neighbour as True in visited dictionary
19:            store new_path in path_nearest_dictionary[neighbour]
20:   end
```

Time Complexity

As line 11-12 and 14-19 run in O(1) time, we are considering them as 1 key operation each.

Pre-processing time:

$Create\ dictionaries\ =\ O(2|V|)\ =\ O(|V|)$

$Append\ hospital\ nodes\ to\ queue\ =\ O(|H|)$

$Pre-processing\ Time\ Complexity,\ P\ =\ O(|V|)\ +\ O(|H|)\ =\ O(|V|\ +\ |H|)\ =\ O(|V|),\ as\ |H|\ \leq\ |V|$

Best Case: Hospital nodes are disconnected from rest of graph	**Worst Case: The graph is a complete graph**
$Number\ of\ iterations\ of\ the\ queue\ =\ \|H\|$ $Number\ of\ edges\ adjacent\ to\ hospital\ node\ =\ 0$ $Number\ of\ key\ operations = 1 * \|H\| = O(\|H\|)$ $Search\ Time\ Complexity,\ S\ =\ O(\|H\|)$ $Total\ Time\ Complexity\ =\ P\ +\ S$ $=\ O(\|V\|)\ +\ O(\|H\|)$ $=\ O(\|V\|\ +\ \|H\|)$ $=\ O(\|V\|),\ as\ \|H\|\ \leq\ \|V\|$	$Number\ of\ iterations\ of\ the\ queue\ =\ \|V\|$ $Number\ of\ edges\ at\ each\ vertice,\ \|E_{adj}\|\ =\ \|V\|-1$ $Number\ of\ key\ operations\ =\ \|V\|\ *\ (1\ +\|E_{adj}\| * 1\ )$ $= \|V\|\ +\ \|V\|\ *\ \|E_{adj}\|$ $=\ \|V\|\ +\ 2\ *\ \|E\|$ $=\ O(\|V\|\ +\ \|E\|)$ $Search\ Time\ Complexity,\ S\ =\ O(\|V\|\ +\ \|E\|)$ $Total\ Time\ Complexity\ =\ P\ +\ S$ $=\ O(\|V\|)\ +\ O(\|V\|\ +\ \|E\|)$ $=\ O(\|V\|\ +\|E\|)$ $=\ O(\|V\|\ +\|V\| * (\|V\| - 1))$ $=\ O(\|V\|^2)$

Average Case: Assume any arbitrary node chosen can be traced back to a specific node.

$Number\ of\ iterations\ of\ the\ queue\ =\ |V|,\qquad Avg.\ number\ of\ edges\ at\ each\ vertice,\ |E_{adj}|\ = 2 * \frac{|E|}{|V|}$

$$\begin{aligned}
Number\ of\ key\ operations &= |V| * (1 + |E_{adj}| * 1\ ) \\
&= |V| + |V| * |E_{adj}| \\
&= |V| + 2|E| \\
&= O(|V| + 2\,|E|)
\end{aligned}$$

$Search\ Time\ Complexity,\ S\ =\ O(|V| + |E|)$

$$\begin{aligned}
Total\ Time\ Complexity &= P + S \\
&= O(|V|) + O(|V| + |E|) \\
&= O(|V| + |E|)
\end{aligned}$$

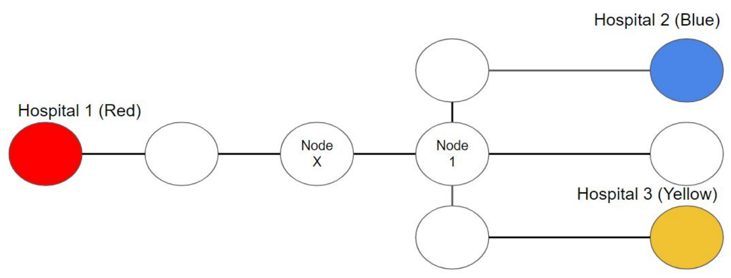
Requirement (c) & (d): Graph in Adjacency List Representation

```
1:   function bfs(graph, hospitals_list, k=no_of_nearest_hospitals )
2:   begin
3:   create dictionary to path_nearest
4:   queue ← emptylist
5:   for each hospital in hospitals_list
6:      queue append list of [current_node, hospital_node, distance_to_hospital]
7:      path_nearest[hospital][hospital] ← 0
8:   while (queue is not empty)
9:      path ← dequeue first list in queue
10:     curr_node ← first item in path
11:     for neighbour in graph[curr_node]
12:        if length of path_nearest dictionary of neighbour >= k
13:           continue
14:        hosp_node ← second item in path
15:        if hosp_node in path_nearest of neighbour
16:           continue
17:        new_path ← list(path)
18:        new_path[0] ← neighbour
19:        new_path[2] ← new_path[2] + 1
20:        enqueue new_path to end of queue
21:        path_nearest[neighbour][hosp_node] ← new_path[2]
22:   end
```

Time Complexity

Lines 9 and 10 run in O(1) time, so we are considering them as 1 key operation. Similarly, line 12 to 21 all run in O(1) time, hence we are also considering them as 1 key operation.

Hospital 1 would traverse to Node X, while Hospital 2 and 3 would traverse to Node 1.The path from Hospital 1 tries to traverse to Node 1, it finds that there are already k paths stored. As we are using breadth-first-search on an undirected graph, we are sure that those are the kth shortest path. As such, all neighbours of Node 1 would have a shorter path to a nearby hospital as compared to hospital 1. Conversely, as the paths from hospital 2 and 3 tries to traverse to Node X, the one first in the queue will traverse down the graph and the other path will terminate. This ensures that all vertices are visited k times.



When we transverse through to the child node, we would check if the child node has >= k. This means if the child node has >= k, we would not be going to the child node as the child node and the nodes in that direction have been reached by k hospitals. Hence we would be iterating through k*|V| times.

Pre-processing time:

$Create\ dictionaries\ =\ O(2|V|)\ =\ O(|V|)$

$Append\ hospital\ nodes\ to\ queue\ =\ O(|H|)$

$Pre-processing\ Time\ Complexity,\ P\ =\ O(|V|)\ +\ O(|H|)\ =\ O(|V|\ +\ |H|)\ =\ O(|V|),\ as\ |H|\ \le\ |V|$

Best Case: Hospital Nodes are disconnected from rest of graph	**Worst Case: The graph is a complete graph**
$Number\ of\ iterations\ of\ the\ queue\ =\ \|H\|$ $Number\ of\ edges\ adjacent\ to\ hospital\ node\ =\ 0$ $Number\ of\ key\ operations = 1*\|H\| = O(\|H\|)$ $Search\ Time\ Complexity,\ S\ =\ O(\|H\|)$ $Total\ Time\ Complexity\ =\ P\ +\ S$ $=\ O(\|V\|)\ +\ O(\|H\|)$ $=\ O(\|V\|\ +\ \|H\|)$ $=\ O(\|V\|),\ as\ \|H\|\ \le\ \|V\|$	$Number\ of\ iterations\ of\ the\ queue\ =\ k*\|V\|$ $Number\ of\ edges\ at\ each\ node,\ \|E_{adj}\|\ =\ \|V\|-1$ $\#\ key\ operations = (k*\|V\|)*(1+\|E_{adj}\|*1)$ $= k*(\|V\|+\|V\|*\|E_{adj}\|)$ $= k*(\|V\|\ +\ 2*\|E\|)$ $=\ k*(\|V\|\ +\ \|E\|)$ $Search\ Time\ Complexity, S = O(k*(\|V\|\ +\ \|E\|))$ $Total\ Time\ Complexity\ =\ P\ +\ S$ $=\ O(\|V\|)\ +O(k*(\|V\|\ +\ \|E\|))$ $=\ O((k+1)*\|V\|\ +k*\|E\|)$ $= O(k*\|V\|\ +\ k*\|E\|)$ $=\ O(k*(\|V\|\ +\ \|E\|))$ $=\ O(k*(\|V\|\ +\ \|V\|\ *(\|V\|-1)))$ $=\ O(k*\|V\|^2)$
Average Case: Assume any arbitrary node chosen can be traced back to a specific node.	
$Number\ of\ iterations\ of\ the\ queue\ =\ k*\|V\|$ $Avg.\ \#\ of\ edges\ at\ each\ vertice,\ \|E_{adj}\|\ = 2*\frac{\|E\|}{\|V\|}$ $\#\ of\ key\ operations\ =\ (k*\|V\|)*(1+\|E_{adj}\|*1)$ $= k*(\|V\|+\|V\|*\|E_{adj}\|)$ $= k*(\|V\|+2\|E\|)$ $=\ k*(\|V\|+\|E\|)$ $Search\ Time\ Complexity, S =\ O(k*(\|V\|+\|E\|))$	$Total\ Time\ Complexity,\ T\ =\ P\ +\ S$ $T\ = O(\|V\|)+O(k*(\|V\|\ +\ \|E\|))$ $= O((k+1)*\|V\|\ +k\|E\|)$ $= O(k*\|V\|+k\|E\|)$ $= O(k*(\|V\|+\|E\|))$

Empirical Analysis

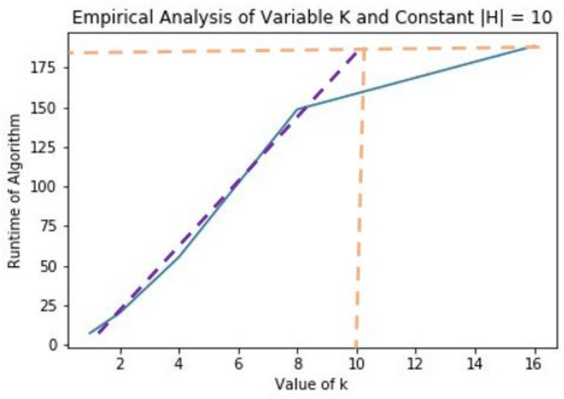
Study Setup

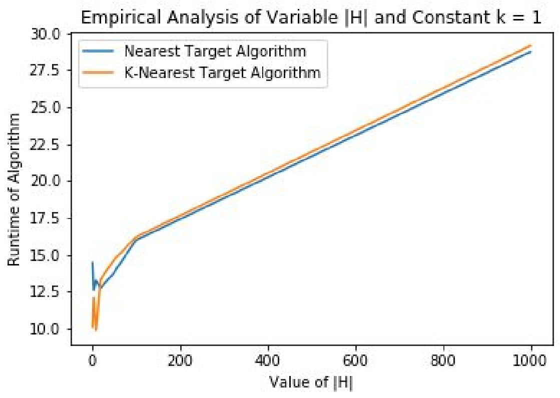
Independent variables: Set of hospital nodes, H, Number of shortest paths from the top, k
Dependent variables: Time
Confounding Variable: Connectedness of the selected hospital nodes

To reduce the effect of the confounding variable, we will run each iteration 5 times on randomly sampled hospital nodes and average the results. Latin Square counterbalance is used.

Constraints	Hold |H| constant, vary k	Hold k constant, vary |H|
Variables	|H| = 10, k values = [1,2,4,8,16]	k = 1, |H| values = [2, 5, 10, 20, 50, 100, 1000]
Expected	Linear graph with positive gradient as time complexity is O(k * (|V| + |E|)). We would expect runtime to increase linearly with respect to k where k <= |H|. When k > |H|, the runtime should show minimal/no increase in run time.	Linear graph with gradient nearly 0 as time complexity is independent of H. Both algorithms should perform nearly the same.

Results





In our variable k graph, when extending the trend line from k = 1 to k = 8, we observe that there is a linear relationship. At k = 10, the predicted run time is the same as the actual runtime when k = 16, confirming our expectations.

References

Narutouzumaki696 (2020, April 24). Multi Source Shortest Path in Unweighted Graph. Retrieved October 20, 2020, from https://www.geeksforgeeks.org/multi-source-shortest-path-in-unweighted-graph/

Vaibhav, J. (n.d.). Flood-fill Algorithm Tutorials & Notes: Algorithms. Retrieved October 20, 2020, from https://www.hackerearth.com/practice/algorithms/graphs/flood-fill-algorithm/tutorial/

Esposito, F. (n.d.). CSCI-1080 Intro To CS: World Wide Web, Graph Theory. Retrieved October 20, 2020, from https://cs.slu.edu/~esposito/teaching/1080/webscience/graphs.html

Statement of contribution

There are three main parts to the project, Implementation of Floodfill BFS for part a) and b), Adapted Floodfill for part c) and d) and the Empirical Analysis for the algorithm in both portions. Brandon and Matthew focused on the design, implementation and time complexity of the algorithms. Keith, Xianan and Stefanie worked on designing the report, empirical study, output formatting and presentation.