NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

# CZ2001 Algorithms

## Project 2: Graph Algorithms

**Lab Group: SS3**

**Group Members:**

| |
|---|
| GOH CHEN KANG, SEAN |
| LEE KAI EN |
| LEE YANG FENG |
| FAVIAN CHAN JUN WEI |
| LIM QI WEI |

## **Introduction**

The aim of this project is to propose and implement an algorithm for finding the shortest paths in a graph. Considering a graph with *n* nodes and *m* edges, there are *h* nodes that are marked as target nodes, the goal is to find the top-k shortest paths from all nodes to the target nodes. For finding the single shortest path, the time complexity of the algorithm shall not depend on *h*. So, we used a multi-source BFS. Programming language used is Java. And, since the real road networks have huge numbers of nodes, we have decided to use adjacency lists to represent our graphs.

## **Algorithm**

### 1. Multi-Source Breadth First Search (BFS)

*Reference: https://www.geeksforgeeks.org/multi-source-shortest-path-in-unweighted-graph/*
*This algorithm above was used as a reference and was modified to meet the project requirements of (a) - (d).*

#### Steps (How it works)

1. Each vertex will be looped through and from each vertex, we run a BFS to find the nearest hospital from that vertex.
2. Before running the algorithm, the following are initialized:
    a. Set indexes of the source hash array where a hospital exists to 1 while the others remain as 0.
    b. Reset all values in the visited array to 0.
    c. Creating ArrayLists for each index of dist and storedPath.
    d. Values in prev array, which stores previous nodes visited, set to -1 which means no parent node.
3. Upon entering the algorithm, a queue is implemented. Before we start pushing anything to the queue, the queue is first cleared. Then, the starting vertex is pushed onto the queue.
    a. The queue holds Pairs, of the form <vertex, dist>, which store the index of the vertex and the distance it is from the hospital, which are retrieved and indicated by variables s and d.
4. If the current vertex has a hospital, the algorithm reconstructs the path and adds the following to their respective lists:
    a. Distance from starting node, d into the List<Integer>[] dist
    b. Shortest path from starting to hospital node, path into List<String>[] storedPath

   Then, it checks if the size of the array stored at the current vertex is equal to k. If it is equal to k, the search stops and exits. Otherwise, it will continue the search for the next closest hospital node.

5. For other vertices with no hospitals, the adjacent unvisited vertices are pushed onto the queue while incrementing value of d, which holds the distance from the starting vertex to each traversed node, and setting the corresponding value in visited[] to 1. At the same time, we will store the previous node in the prev[] array to be used later.
6. If a hospital node is reached, step 4 is repeated and size of the array is checked again. It exits if it is equal to k. Otherwise, it continues.
7. Steps 4 - 6 are repeated while the queue still has more unvisited vertices.

## Algorithm Design Assumptions

1. When finding the top k nearest hospitals, we assume that we are still able to go through the vertices, which were already traversed and are hospital vertices, to search for the next nearest hospital vertex instead of having to go around the vertex.

## Time Complexity Analysis

- Let V be the number of vertices, E be the number of edges of the current vertex, k be the number of nearest paths and the time cost of each loop be c.

### Best Case

- The case where there are no hospitals will not be discussed because there is no point in searching if it is known that there are no hospital vertices.
- Best case occurs when all vertices are sources which means there is a hospital in all the vertices which causes the BFS to end before reaching the for loop and that k is less than the number of edges at the current vertex, for top k nearest hospitals .
- For k = 1, it means that there will only be 1 loop per vertex while looping through the vertices.
- And, for k nearest hospitals, it means that the algorithm will loop an additional k times since its immediate neighbouring nodes are hospitals and needs k loops to dequeue them, for k <= E.
  - If k <= E, time cost will be $V \times [(1+k) \times c]$
- Assuming equal probability for each case, this gives a time cost of $V \times [(1+k) \times c] \rightarrow$ time complexity = $O(V + Vk) = O(Vk)$

## Worst Case

- The worst case will be when there is only 1 hospital and so the only possible value of k is 1 since the program only allows an input, ranging from 0 to the number of hospitals (inclusive).
    - The vertices closest to the hospitals will have a time cost of 1*c since they exit the search in the 2nd iteration of the while loop. Following that, the next closest would have time cost of 2*c and the next will be 3*c, etc.
    - Assuming that all vertices are reachable from the starting node,
      Time cost
      = $(1 \times c) + (2 \times c) + (3 \times c) + ... + [(V-3) \times c] + [(V-2) \times c] + [(V-1) \times c]$
      = $c \sum_{i=1}^{V-1} i$
      = $c[1 + 2 + 3 + ... + (V-3) + (V-2) + (V-1)]$
      = c ( $\frac{V(V-1)}{2}$ ) → time complexity = **O(V$^2$)**

## Average Case

- For k = 1,
    - Assuming all vertices are reachable from the starting vertex, and probability for each case is equal,
    - Different cases include having 1 hospital vertex, 2 hospital vertices, 3 hospital vertices, … , V-2 hospital vertices, V-1 hospital vertices, V hospital vertices.
    - Total no. of cases = V → P(each case) = $\frac{1}{V}$
    - Time cost
        - = $\frac{1}{V}(\frac{V(V-1)}{2}c) + ... + \frac{1}{V}((V-1)c + 2c) + \frac{1}{V}(Vc)$
        - = $c(\frac{1}{V})[(\frac{V(V-1)}{2}) + ... + ((V-1) + 2) + V]$
          → since time cost is highest when there is only 1 hospital vertex, and
          → time cost decreases as we go down to V hospital vertices,
          → ranging from a quadratic expression, V$^2$ to just V, hence
          → general form of total time cost = $c(\frac{1}{V})(aV^2 + bV + d)$
          → where a, b and d are constants and a $\neq$ 0, b $\neq$ 0
    - Time complexity
        - = $O(c(\frac{1}{V})(aV^2 + bV + d))$
        - = $O(aV + b + \frac{d}{V})$
        - = $O(V + \frac{1}{V}) = O(V)$

## Space complexity

Considering the worst case:

- source hash array → $O(V)$
    - 1D array with v vertices
- visited array → $O(V)$
    - 1D array with v vertices
- prev array that stores path → $O(V)$
    - 1D array with v vertices
- dist array → $O(kV)$
    - 2D array of dimensions $k \times v$
- storedPath array → $O(kV)$
    - 2D array of dimensions $k \times v$
- Dequeue → $O(V)$
    - as all vertices are in the queue

Total space complexity = $O(V + V + V + kV + kV + V) = O(4V + 2kV) = O(V + kV)$

## Empirical Analysis

What happens to the execution time of the program as we vary the h and k?
Values recorded are in seconds, s.
A randomly-generated graph with 2000 nodes was used to measure the execution times below.

| h/k | 1 | 10 | 100 |
|---|---|---|---|
| 1 | 0.3672235 | - | - |
| 10 | 0.3736014 | 0.6422324 | - |
| 100 | 0.0470528 | 0.4678021 | 0.8419961 |
| 1000 | 0.0065562 | 0.0661051 | 0.3434869 |

Trend

- For the same value of h, as k increases, execution time increases. This is because more hospital vertices have to be explored hence running time increases and each increase from k = 1 to k = 100 has been more than 2 times of its previous execution time.
- For the same value of k, as h increases, execution time decreases. This is because there is a greater probability that a neighbouring vertex is a hospital vertex, making the search faster. And, each increase of h from 10 to 100 to 1000 has been met with more than 2 times increased speed

For real road networks, realNet-CA.txt was used and for h = 1,000,000 & k = 1, execution time = 1 hour 17 minutes as evidenced by results shown in the realNet-CA_results 1 folder

## Statements of Contribution

GOH CHEN KANG, SEAN:
- Helped out in algorithm formulation for part (a) & (b)
- Space complexity analysis
- Slides

LEE KAI EN:
- Converted the C++ code from our reference to Java
- Modified the code to meet project requirements of parts (c) & (d), top k shortest paths to hospital
- Created a UI, added code to store/read randomly generated graph and hospital nodes to/from txt files to be used as input files.
- Merged and debugged the complete program
- Time complexity analysis of the algorithm and report

LEE YANG FENG:
- Research of graph algorithm

FAVIAN CHAN JUN WEI:
- Searched for algorithms to be used as references
- Helped out for parts (a) & (b)

LIM QI WEI:
- Helped out with path reconstruction, graph reading and hospital generation inputs required for road network data using a self constructed graph.
- Helped with compilation of various code files from different people.