NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Project 2: Graph Algorithms

02/11/2020

Lab Group: SE1

Submitted by:
Siddharth M Sundaram
Agnesh Ramesh
Aditya Chandrasekhar
Moshik Seetloo

Note: Number of nodes = V, Number of edges = E, Number of hospitals = H

1. Breadth-first search (optimized)

Idea: Traverse using BFS from every hospital, sequentially, exploring the neighbouring nodes. **Optimisation:** If the distance from the current hospital to a node is greater than the current minimum distance of the node, the entire branch will be ignored by the algorithm.

Pseudocode:

```
function bfs_optimized(adj, source, previous, curr_min, hospital)
begin
    distance of source ← 0
    add source to queue
    curr_min of source ← (distance of source, hospital)
    while queue not empty do
        u ← 1st element of queue
        pop 1st element
        for each vertex v adjacent to u do
            if distance of v less than curr_min of v and v is not visited
                add v to queue
                distance of v ← distance of u  +1
                previous of v ← u
                curr_min of v ← (distance of v, hospital)
    return distance, previous, curr_min
end
```

The above code is called for each hospital, and the curr_min and prev arrays provide the required information.

Proof of Correctness: Assume that while performing this operation for a hospital H, at a node X, the distance d from H is greater than the current minimum distance m from hospital H'. As a result node X and all its adjacent nodes will not explored. We know that $m < d$. All nodes adjacent to node X have distance $(d + 1)$. Now, all these nodes would be at distance $(m + 1)$ from H'. Since $(m + 1) < (d + 1)$, it is not possible to find a shorter distance from H to the nodes adjacent to node X. This argument can now be extended to all nodes at distance 2, distance 3 and so on, from node X. Therefore, this algorithm leads to the shortest path.

Time Complexity:

Worst Case: Happens when the last hospital is the nearest to all the nodes hence BFS has to be performed H number of times. Therefore:

$$F(V, E, H) = H \times (V + E) = O(H.(V + E)$$

Average Case: Assume on average, that the set of V nodes can be divided into H sets of V/H nodes, such that each set has a common nearest hospital (i.e. H hospital trees). Then, BFS has to be performed completely for the first node, while for the second node, the branch of nodes which have minimum distance to first hospital will be excluded, and for the third node, the branches of nodes which have minimum distance to first and second hospitals will be excluded, and so on.

$$F(V, E, H) = (V+E)+[(V+E) - (V+E)/H]+[(V+E) -2(V+E)/H]+...+[(V+E) -(H-1)(V+E)/H]$$
$$= [H.(V + E) – (V + E)/H.(1 + … + (H – 1)] = H.(V + E) – (V + E)(H – 1)/2 = O(H.(V + E)$$

Best case: Happens when the first hospital is the nearest to all the nodes. Hence BFS is done only once, since all branches will be excluded for all other hospitals. Therefore:

$$F(source) = 1x(V+E) = O(V+E)$$

Space Complexity: Two additional lists were used, **curr_min** that stores the nearest hospital for all V vertices and **prevs** that is used to retrieve the shortest path to the nearest hospital from all V vertices. Therefore:

$$\text{Space complexity} = V.H+V = O(V.H)$$

2. Multi source Breadth-first search[1]

Idea: To perform BFS from multiple sources simultaneously, one layer at once to find the shortest distance to the nodes.

Pseudocode:

```
function bfs_hospital_wise(adj, hospitals)
        dist ← dictionary to store distance and nearest hospital
        mark all vertices as unvisited
        current hospital tree ← current hospital
        add all hospitals to queue
        for every h in hospitals do
                mark h as visited
                dist of h  ← [0, h]
                while queue not empty do
                        u, curr_h ←  1st element of queue
                        pop 1st element
                        for each vertex v adjacent to u do
                                if v is not visited
                                        mark v as visited
                                        add v to queue
                                        dist of v ←  [dist of u + 1, curr_h]
                                        add edge uv to current hospital tree
        return dist, tree
end
```

Proof of Correctness: The algorithm uses all the hospital nodes as starting points, and does a simultaneous BFS on each hospital. The BFS for each hospital will only visit nodes that have not been traversed yet. This will partition the graph into clusters of nodes nearest to each hospital, as a BFS for one hospital will not overlap the BFS for another hospital. The nodes in each cluster will be discovered in the respective hospital's BFS first (due to the nature of BFS), and will therefore lead to the shortest path.

Time Complexity: In the average, worst and best cases, each vertex V and each edge E is only visited and traversed once. Therefore:

$$F(V, E, H) = 1x(V+E) = O(V+E)$$

Space Complexity: Three additional dictionaries were used, **dist** that stores nearest hospital and its distance for all V vertices, **visited** that marks when a vertex V is visited and **prev** that is used to retrieve the shortest path from all V vertices to all hospitals H. Therefore:

$$\text{Space complexity} = V + V + V.H \approx \Theta(V.H)$$

3. Top-k Nearest hospital

Idea: Find the closest nodes from each of the hospitals, thereby also finding the shortest distances to each of the hospitals from all of the nodes. Retrieve top-k nearest hospitals for each node.

Pseudocode:

(Note that bfs below refers to normal BFS implementation using queue[2])

```
function compute_top_k(adj, hospitals, k):
        distance_hospital ← dictionary to store distance of a vertex from each hospital
        for every h in hospitals:
                dist ← bfs(adj, h, prev)
                distance_hospital[h] ← dist
        distance_node ← dictionary to store hospital and its distance from each vertex
        distance_node = transpose(distance_hospital)
        for every list in distance_node
                list ← first k elements of sorted(list)
        return distanceFromEachNode
end
```

Proof of Correctness: Once the shortest distance from a node to all the hospitals has been determined, we can easily find out the top **k** closest hospitals to a given node by arranging the distance array in ascending order and truncating it to the first **k** elements.

Time Complexity: In the average, worst and best cases, each vertex V and each edge E is visited or traversed once for every hospital. There is also quicksort done at the end of the traversal to arrange the k values in ascending order.

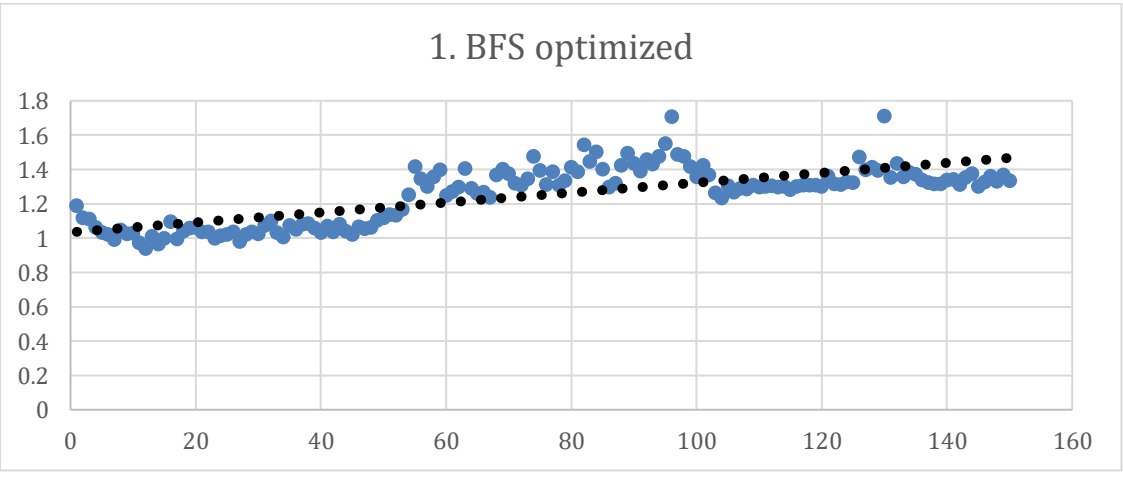
$$F(V, E, H, k) = H(V+E) + V.H + (H\log H).V = O((V + E).H)$$

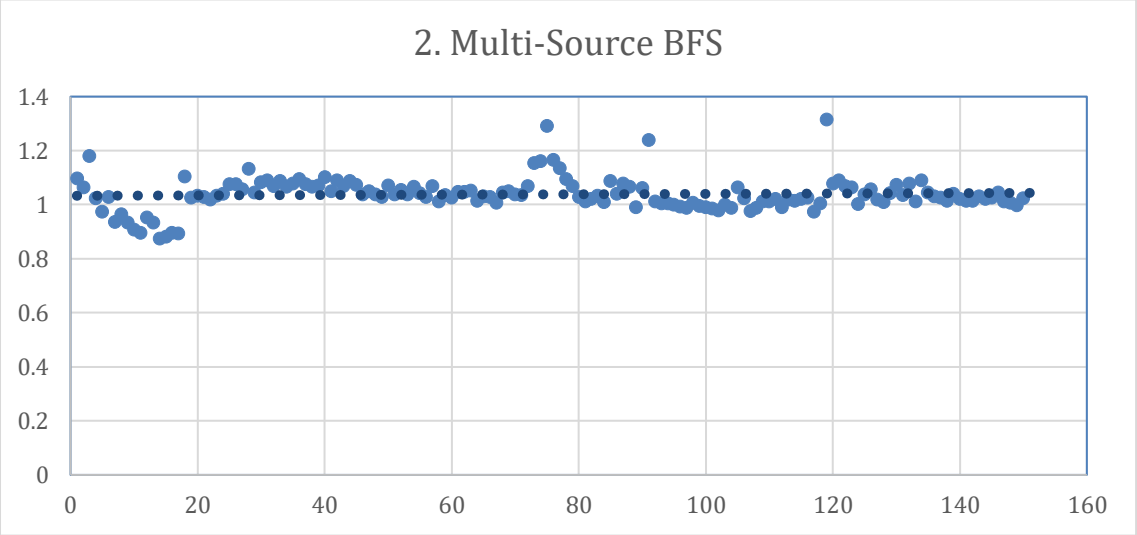
Space Complexity: In normal BFS, **dist** is used that stores nearest hospital and its distance for all V vertices, **prev** that keeps track of the path (hospital tree). Further, 2 extra dictionaries **distance_hospital** and **distance_node** is also required to keep track of the distances of each of the hospitals to every node and vice versa.

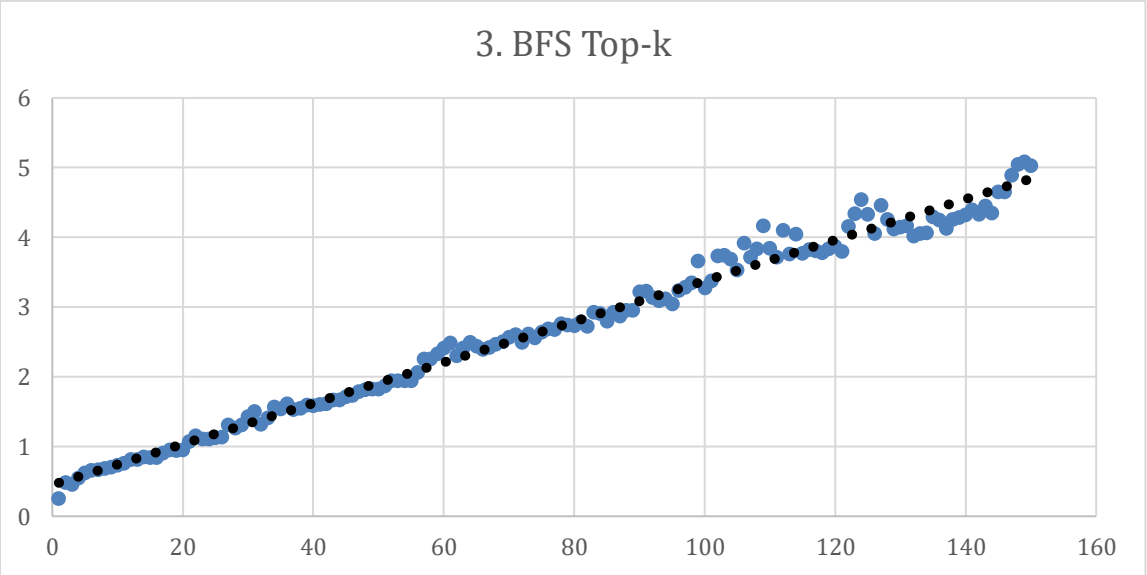
$$\text{Space complexity} = V + V + V.H \approx \Theta(V.H)$$

Observations and Conclusions:

On performing empirical analysis on a random graph with 10000 nodes and 15000 edges as the number of hospitals increases, the following 3 plots were obtained. It can be clearly seen that the 2nd algorithm (used for part (b)) is independent of the number of hospitals.







References

[1] https://www.geeksforgeeks.org/multi-source-shortest-path-in-unweighted-graph/
[2] https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/