CZ2001 ALGORITHMS

Project 2: Graph Algorithms

Bairi Sahitya
Loh Xin Yi
Satini Sankeerthana
Shauna Tan
Ong Li Wen

Algorithm analysed:
Multi Source Breadth-First Search

CZ2001 - ALGORITHM PROJECT 2

Finding the nearest hospital using Multi Source Breadth-First Search (MS BFS)

MS BFS is built on the BFS algorithm where we have multiple BFS traversals occurring concurrently. In MS BFS instead of pushing only one node into the queue, all the sources are pushed into the queue first. After which, the BFS traversal proceeds as per normal. Initially it marks the source of each BFS, it then adds all the source nodes (hospital nodes) into the queue. Then from each BFS level, the algorithm uses the queue to repeatedly select and find which node is to be explored.

When a node is removed, its unexplored neighbours are marked and pushed into the queue. A distance and a predecessor array is updated every time a node is pushed into the queue. Each of the indexes of the distance and predecessor array match the node IDs. The distance array stores the distance to the nearest hospital for each node while the predecessor array stores the node ID of the predecessor node in the nearest path for each of the nodes. E.g.
distance[6] = the distance to the nearest hospital from node 6.
parent[6] = node ID of the predecessor to node 6 in the shortest path.

Figure 1: MS BFS Code to compute the nearest hospital:

```
public static void multibfs(ArrayList<vertex> hospitals, ArrayList<LinkedList<vertex>> adjacencyList)
{
    Queue<vertex> q=new LinkedList<vertex>();

    for (int i=0; i < hospitals.size(); i++ )
    {
        hospitals.get(i).mark();
        dist[hospitals.get(i).getLabel()] = 0;
        parent[hospitals.get(i).getLabel()] = -1;

        q.add(hospitals.get(i));
    }

    while(!q.isEmpty())
    {
        vertex n=(vertex)q.remove();
        LinkedList<vertex> vertexLinkedList = adjacencyList.get(n.getLabel());
        for (vertex neighbour : vertexLinkedList) {
            if(!neighbour.isMarked()) {
                neighbour.mark();
                dist[neighbour.getLabel()] = dist[n.getLabel()] +1;
                parent[neighbour.getLabel()]= n.getLabel();
                q.add(neighbour);
            }
        }
    }
}
```

Thus the distance to the nearest hospital can be directly taken from the array. The shortest path can be computed by recursively checking the predecessor to a node until a hospital is reached.

Figure 2: Code to print the shortest path:

```
public static void printpath(int vertex) {

    if (vertex == -1) {
        System.out.println(" -This is the nearest hospital");
    }
    else if (parent[vertex] == -99) {
        System.out.println("Such a node id does not exist");
    }
    else {
        if (parent[vertex] == -1) {
            System.out.print(vertex);
        }
        else {
            System.out.print(vertex + " --> ");
        }
        printpath(parent[vertex]);

    }
}
```

Figure 3: Example Output (where 1 is the nearest hospital to node 13):

```
The distance to the nearest hospital from node 13 is: 4
13 --> 10 --> 6 --> 2 --> 1 -This is the nearest hospital
```

Time Complexity Analysis

In our analysis of the time complexity, only the time taken for the mulitbfs() function in Figure 1 is accounted for. The time taken to print the shortest path (ie: printpath() function in Figure 2) is not accounted for.

In the MS BFS, each edge is processed once in the while loop for a total cost of $\Theta(|E|)$ where E is the number of edges. Each node is queued and dequeued once for a total cost of $\Theta(|V|)$. Since, we have used an adjacency list to represent our graph, O(1) time is taken to get the neighbours at each node. Thus, the Worst-case time complexity of the MS BFS is $\Theta(|V|+|E|)$. The MS BFS is independent of the number of hospital nodes, as each BFS traversal includes visiting each node and edge only once regardless of the number of hospital nodes.

Modified Multi Source BFS algorithm to find nearest k hospitals

The algorithm used to find the nearest k hospitals builds on the one used to find the nearest hospital with 4 main modifications.

Firstly, at each iteration instead of a node being pushed into the queue, a pair of vertices are pushed into the queue. The first node in the pair is the node to be explored while the second node is the nearest source (hospital node).

Secondly, for each of the node-source pair, the pair is added to the queue if the node is visited less than k times and if the source is not among the nearest hospitals already computed for that node. When the shortest distance from a node to a particular hospital is computed, the hospital is marked within the node object.

Thirdly, the structure of the distance and the parents arrays is modified to store more information. Distances is now a two dimensional array which contains k distance arrays. Hence, the nth row has the distance to the nth nearest hospital from each of the nodes. The Predecessor or parent array is now a three dimensional array where
parents['k']['node ID of predecessor']['row of the predecessor node to continue path from']

Lastly, an attribute called nRemoved is tracked for each of the node objects. It is incremented every time a node is removed from the queue. This allows the algorithm to get the row of the predecessor to add to the node and the respective distance to use.

Figure 4: Code to find the nearest k hospitals

```
public static void multibfs_k(ArrayList<vertex> hospitals, ArrayList<LinkedList<vertex>> adjacencyList, int k)
{
    Queue<vertex[]> q=new LinkedList<vertex[]>();

    for (vertex i : hospitals)
    {
        i.incVisited();
        i.addSource(i.getLabel());
        parents[0][i.getLabel()][0] = -1;
        vertex[] pairs = {i,i};
        q.add(pairs);
    }
    while(!q.isEmpty())
    {
        vertex[] v=(vertex[])q.remove();
        vertex n = v[0];
        vertex s = v[1];
        n.incRemoved();
        LinkedList<vertex> vertexLinkedList = adjacencyList.get(n.getLabel());
        for (vertex neighbour : vertexLinkedList) {
            if(neighbour.getnVisited()<k && !neighbour.checkSource(s.getLabel())) {
                neighbour.incVisited();
                neighbour.addSource(s.getLabel());
                distances[neighbour.getnVisited()-1][neighbour.getLabel()] = distances[n.getnRemoved()][n.getLabel()] +1;
                parents[neighbour.getnVisited()-1][neighbour.getLabel()][0]= n.getLabel();
                parents[neighbour.getnVisited()-1][neighbour.getLabel()][1]= n.getnRemoved();
                vertex[] pairs = {neighbour, s};
                q.add(pairs);
            }
        }
    }
```

Time complexity of modified algorithm

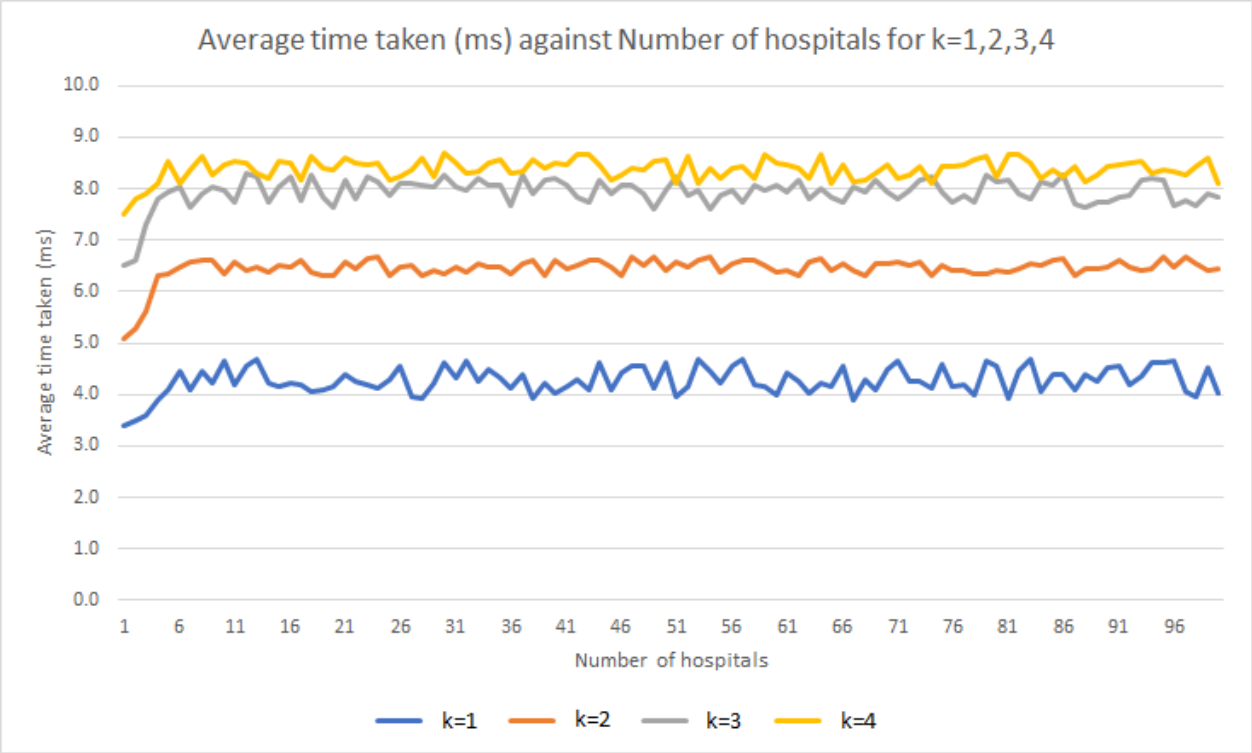
Using the modified algorithm, each node is queued and dequed k times and each edge is visited exactly k times. The additional steps of marking a computed source for a vertex or incrementing the number of times it is visited can all be done in O(1) time due to the way the attributes are implemented in the vertex class in java. Thus the time complexity for this algorithm is $\Theta(k(|V|+|E|))$.

Empirical Analysis

To further analyse the time complexity of the algorithm, an empirical study was conducted using the same graph with 1000 nodes for all measurements and the same set of hospitals for each number of hospitals across all levels of k. Firstly, to determine if the number of hospitals, h, in the graph affects the time taken for the MS BFS algorithm to run, the time taken (in ms) was measured for each h between 1 and 100 30 separate times, then the average time taken is calculated for each h. From the graph below, it can be seen that the general trend for each level of k (each line

represents one level of k) are very similar and the average time taken follows a generally constant trend (flat line) as the number of hospitals increases. This proves that the MS BFS algorithm is independent of the number of hospitals. Comparing between different levels of k, it can be seen that from the total average time taken increases as k increases, thus proving that the MS BFS algorithm depends on the value of k. Therefore, the empirical analysis conducted is in line with the time complexity of the MS BFS algorithm, which is $\Theta(k(|V|+|E|))$.

Figure 5: Graph of Average time taken (ms) against Number of hospitals for k=1,2,3,4



References:

https://db.in.tum.de/~kaufmann/papers/msbfs.pdf
https://www.geeksforgeeks.org/multi-source-shortest-path-in-unweighted-graph/

Statement of Contribution:

Development of algorithm to find nearest hospital and get path: Li Wen and Xin Yi

Modification of Algorithm to find nearest k hospitals: Sahitya and Sankeerthana

Reading and writing to files, converting text file to adjacency list, and empirical analysis: Shauna