**Mahlet Zemedie**

**17/11/2018**

**Lab5 Report**

**Introduction**

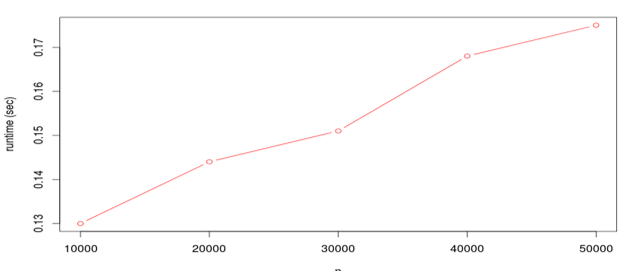
A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. Graphs are used to solve many real-life problems. Graphs are also used in social networks like linkedIn, Facebook. Breath first search (BFS) for a graph is similar to Breadth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex. Breadth-first search (BFS) is an important graph search algorithm that is used to solve many problems including finding the shortest path in a graph and solving puzzle games. Many problems in computer science can be thought of in terms of graphs. For example, analyzing networks, mapping routes, and scheduling are [graph problems](https://brilliant.org/wiki/graphs/). Graph search algorithms like breadth-first search are useful for analyzing and solving graph problems.

**Method**

The other task is to design and implement Breath First Search using recursive and Iterative methods. The methods checked for a variety of graphs like cyclic/acyclic, directed/undirected, having one or more connected components. The plots for the BFS-recursive and BFS-iterative need to be plotted as a function of number of nodes and number of edges. An input file must be scanned, and a graph needs to be created and then the generated graph needs to be written to another file. The input and output graphs are visualized using R. The file is read using the scan method and the graph is created using the graph data structure. The addition of edges is done in constant time by using sets and hash table. The BFS function will take in a graph G = (V,E) and a source node to start the BFS from. The function will return a distance label for each node. The distance label for a given node indicates the distance of the node from the source node. They will be stored in the graph data structure – the graph class from Lab 4 is used, and the class is enhanced. Queue is used to store the discovered but not yet being processed nodes.

**Result**

Five cases are used for testing. The output for each case are plotted above. The graph is implemented using adjacency lists, wherein each node maintains a list of all its adjacent edges, then, for each node, we could discover all its neighbours by traversing its adjacency list just once in linear time. For a directed graph test cases, the sum of the sizes of the adjacency lists of all the nodes is E (total number of edges). So, the complexity of BFS is O(V) + O(E) = O(V + E). For an undirected graph, each edge will appear twice. Once in the adjacency list of either end of the edge. So, the overall complexity will be O(V) + O (2E) ~ O(V + E).



**Discussions**

The graph for the Recursive function where the number of nodes is increasing for a constant value of number of edges is linear. The graph for the Iterative function where the number of nodes is increasing for a constant value of number of edges is linear. The graph for the Recursive function where the number of edges is increasing for a constant value of number of nodes is linear. The graph for the Recursive function where the number of edges is increasing for a constant value of number of nodes is linear.