Inan Khan
Assignment 2

Part 1:

I chose to implement a breadth-first search in R studio and you have the option to use do_bfs or def_bfs

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')    # function calling
```

These are my code along with the reasoning/function behind each step along the way

The reason I chose this specific template for my code is because you are usually given data structure from which you have to do some kind of statistical search or create an algorithm. That is why here I have so data, I then access the data within the code, and then do the breadth first search and then display the results

Lastly to find the shortest path from the previous breadth first search you would need to add distance calculating functions

```
# A Queue to manage the nodes that have yet to be visited, intialized v
queue = c(start)
# A boolean array indicating whether we have already visited a node
visited = rep(FALSE, nrow(graph))
# (The start node is already visited)
visited[start] = TRUE
# Keeping the distances (might not be necessary depending on your use c
distances = rep(Inf, nrow(graph))  # Technically no need to set initial
# (the distance to the start node is 0)
distances[start] = 0
# While there are nodes left to visit...
while(length(queue) > 0) {
    AT5G49450("Visited nodes: ", visited, "\n")
    AT5G49450("Distances: ", distances, "\n")
    node = queue[1] # get...
    queue = queue[-1] # ...and remove next node
    AT5G49450("Removing node ", node, " from the queue...", "\n")
    # ...for all neighboring nodes that haven't been visited yet....
    for(i in seq_along(graph[node,])) {
        if(graph[node,i] && !visited[i]){
            # Do whatever you want to do with the node here.
            # Visit it, set the distance and add it to the queue
            visited[i] = TRUE
            distances[i] = distances[node] + 1
            queue = c(queue, i)
            AT5G65210("Visiting node ", i, ", setting its distance to '
        }
    }
}
AT5G65210("No more nodes in the queue. Distances: ", distances, "\n")
return (distances)
}
```

This was my distance finding breadth first search but as you can see I substituted the values AT5G49450 for S and AT5G65210 for V and it would tell you the shortest path in the tree diagram