FINDING AN OPTIMAL PATH WITH DIJKSTRA'S ALGORITHM

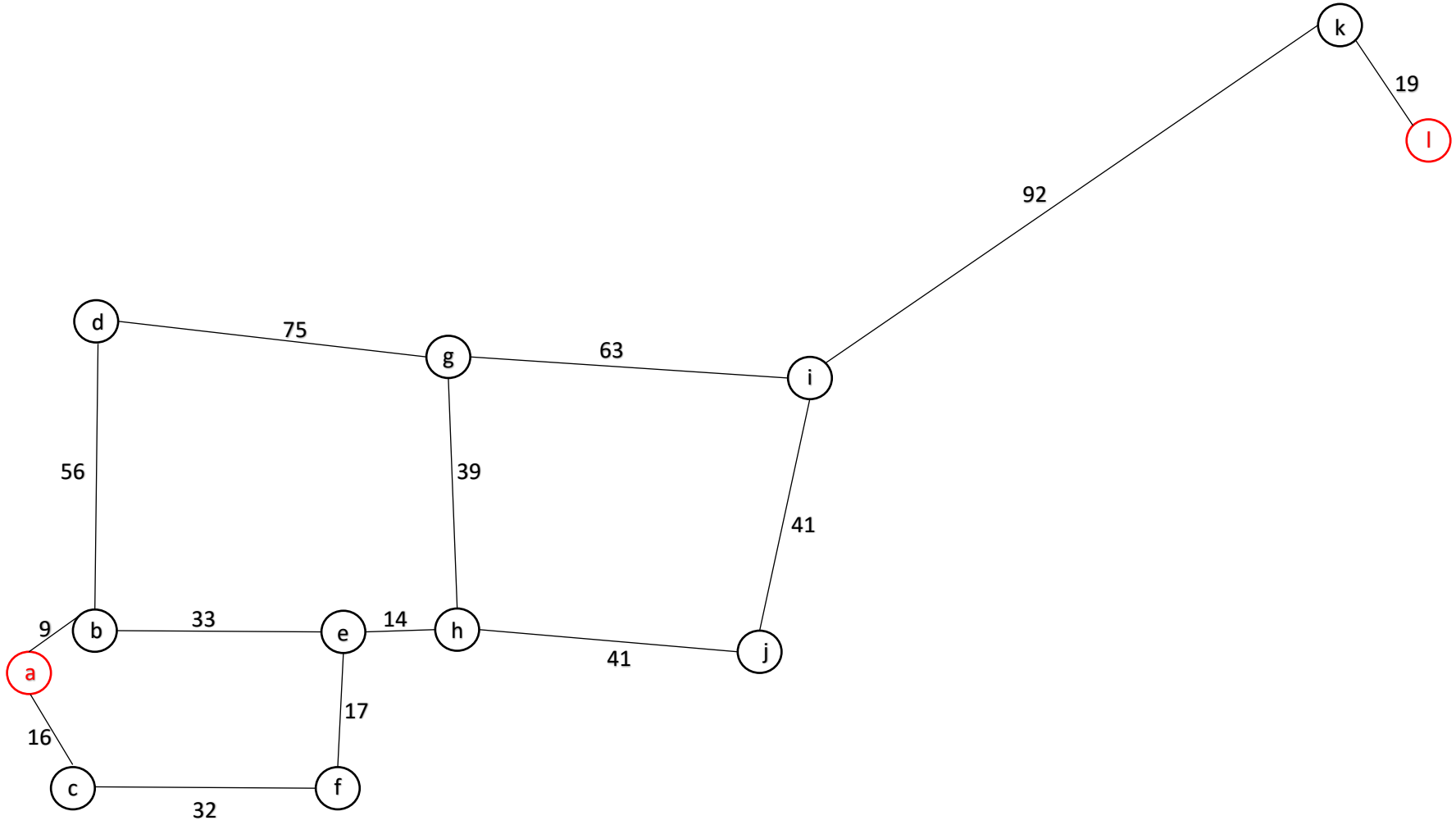
PRESENTED BY – YEAMIN FARIA CHOWDHURY

INTRODUCTION

- Roads are an important part of our day to day life for travelling to work, school or for transportation of goods.
- Manipulation of shortest paths in important locations like, airport, industrial areas is an important concern in this era of rapid urbanization due to over dependency on street transportation. The application of shortest path algorithms in road network is inevitable in emergency handling situations.
- This single source shortest path problem can be addressed using Dijkstra's algorithm with a graph of nonnegative edge path costs.
- Among all other search algorithms to find the shortest path, Dijkstra's algorithm is the fastest, well known and more efficient in terms of computation.

NODES OF STUDY AREA IN GOOGLE EARTH

NODES WITH WEIGHTED EDGES



IMPLEMENTATION

- The study area is an area to the north of Dhaka city in Bangladesh. The shortest optimal path between two nodes ‘a’ to ‘l’ which is from a residential area to the airport is found out.
- The nodes are taken from google earth and the weighted edges are taken. Then these are taken as input in the variable graph of the python code that I implemented.

```
graph = {

'a':{'b':9, 'c':16},
'b':{'e':33, 'd':56},
'c':{'f':32},
'd':{'g':75},
'e':{'f':17, 'h':14},
'f':{'e':17},
'g':{'h':39, 'i':63},
'h':{'j':41},
'i':{'k':92},
'j':{'h':41, 'i':41},
'k':{'l':19},
'l':{'k':19}
}
```

IMPLEMENTATION

- The function Dijkstra passes three parameters – graph, start node and goal node. Then some empty variable is created.
  - The variable ‘shortest_distance’ stores the distance between the nodes.
  - ‘previous_node’ tracks the previous path while reaching new node.
  - ‘track_path’ stores optimal route through the journey from source to goal.

The variable ‘all_nodes’ iterates though all the nodes in the graph and ‘infinite’ is the initial distance between all the nodes before iteration.

IMPLEMENTATION

```
def dijkstra(graph, start, goal):
    shortest_distance = {}      # stores the shortest distance to reach to the code
    previous_node = {}          # tracks previous path while reaching new node
    all_nodes = graph           #iterates through all the nodes in the graph
    infinite = float('inf')
    track_path = []             #stores optimal route through the journey from source to goal

    for node in all_nodes:
        shortest_distance[node] = infinite
    shortest_distance[start] = 0

    while all_nodes:
        min_distance_node = None
        for node in all_nodes:
            if min_distance_node is None:
                min_distance_node = node
            elif shortest_distance[node] < shortest_distance[min_distance_node]:
                min_distance_node = node
        path_options = graph[min_distance_node].items()    # records the path taken for the mean node

        for child_node, weight in path_options:
            if weight + shortest_distance[min_distance_node] < shortest_distance[child_node]:
                shortest_distance[child_node] = weight + shortest_distance[min_distance_node]
                previous_node[child_node] = min_distance_node

        all_nodes.pop(min_distance_node)
    currentNode = goal
    while currentNode != start:
        try:
            track_path.insert(0, currentNode)
            currentNode = previous_node[currentNode]
        except KeyError:
            print("Path is not reachable")
            break
    track_path.insert(0,start)
    if shortest_distance[goal] != infinite:
        print("Shortest distance is: " + str(shortest_distance[goal]))
        print("Optimal path is " + str(track_path))
```

IMPLEMENTATION

- The while loop is created to explore all the nodes in the graph.
- Initially the variable ‘mean_distance_node’ is set to None.
- If the distance of the new node is less than the distance of the node in ‘mean_distance_node’ then it will be updated and will be equal to that new node.
- ‘path_options’ will record the path that is taken for the updated ‘mean_distance_node’.
- If another child node is found which has a lower distance than the particular node, then that will be updated in ‘shortest_distance’ and this will be equal to the summation of the weight and mean_distance_node.
- Now the ‘previous_node’ will track the path that leads to the ‘child_node’.
- In order to stop the while loop, ‘all_nodes.pop’ will be used to pop out the nodes ones after all the nodes have been gone through.

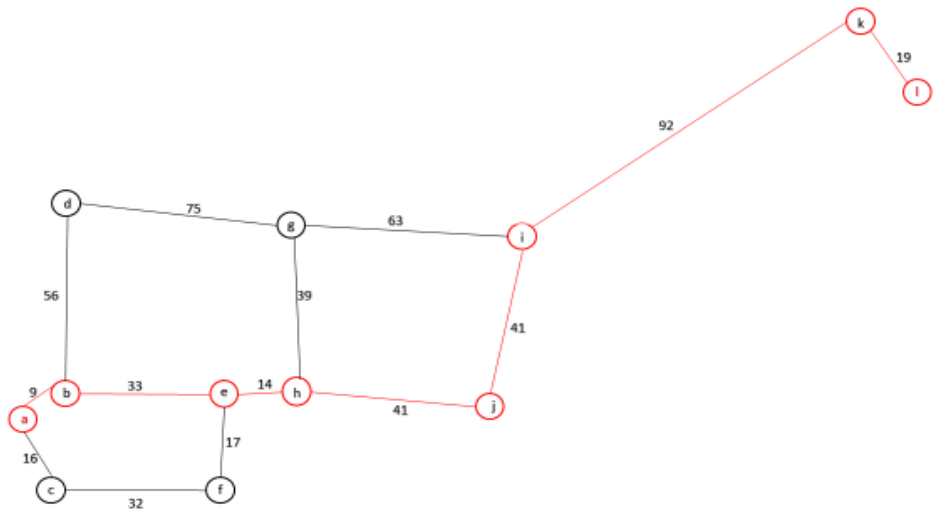
IMPLEMENTATION

- Now, considering ‘currentNode’ as the goal node, all the previous nodes are traced back until the currentNode is not equal to the start node since the start node does not have a previous node. The ‘currentNode’ will be updated in the ‘previous_node’.
- In case there is no path between the start and end node, the program will raise a KeyError and print ‘Path is not reachable’ and break the forloop.
- Now, the start node is inserted by ‘track_path.insert’.
- If the distance up to the goal node is not infinite, then it will print ‘The shortest distance is’ then the distance up to the goal node is shown by ‘shortest_distance[goal]’ and printing ‘Optimal path is’ will show the path shown in the shortest distance.

IMPLEMENTATION

- Calling the function ‘Dijkstra’ will show the following result. And the optimal path on the graph is also shown.

```
Shortest distance is: 249
Optimal path is ['a', 'b', 'e', 'h', 'j', 'i', 'k', 'l']
```



THANK YOU