# **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**IX730151 Year Three Special Topic**

**Task 4**

# **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |
| --- | --- |
| Study Block 3: | 2022 |
| Due date to be handed in: | Tuesday 13th Sept 9 am |
| Submission: | Upload soft copy to Moodle |
| Contact Lecturer: | Tariq Khan |
| Designed by: | Farhad Mehdipour |
| Weighting/Contribution: | 20% to final marks |
| Learning outcomes covered: | 1 |

**Submitted by: Sanim Pokhrel\_**

What is Dijstra’s Algorithm?

Dijkstra’s algorithm helps us to find the shortest path between nodes in a graph from a node (source node) to all other nodes in a graph. This algorithm is most popularly used in GPS systems to find the shortest path between current location and destination.

For the shortest path algorithm I have used a DFS search to find all the possible path from a source node to the destination node .

The paths from starting mode to ending node is stored in a Array List .

A priority queue is used to store the path of the node along with the total weight(cost) of the path.

A Boolean array that stores weather the vertices(nodes) of the graph are visited or not.

The adjacent edges are calculated using the 0.5\*weight(x) <= weight(y) <= 2\*weight(x)

A special node can be added and the application ensures that the shortest path will have the special node.

2

2

2

3

2

Fig.1. showing an example of a undirected weighted graph.

Let us suppose the starting and the ending nodes are A and D respectively. Whilst we traverse through the graph using DFS:

All the nodes with it’s weight have been stored in an adjacency list which provides us the direction for the graph

Every node that has been encountered will be marked visited or true in the array and once we reach and end we back track to the previous node and check if it has an edge with some other node. If true continue searching if not back track again.

When the start node is equal to the end node the search stops and the list and the total cost is added to the priority queue.

The priority queue is responsible to show the lowest cost with it’s path as it’s front element.

For our example of the graph:

2

2

2

In the first search we encounter the above nodes and hence store the total cost from A to D that is 6 and the path followed would be A->B->C->D

The path and the cost will be added to the priority queue.

Similarly all possible nodes in the list are traversed and the path are stored.

Path taken: A->B->D

3

2

Cost: 2+3 =5

Path Taken: A->C->D

2

2

Total cost: 2+2 = 4.

After we find all the path and calculate their cost, we then check if the path has an edge and if it has the special vertex we specified. If true(if it has both vertex and adjacent edge) the we add the path and cost to the priority queue.

At the end the path at the front index of the queue will always have the lowest cost(Weight) with it’s shortest path.