TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING

PURWANCHAL CAMPUS

Dharan-8

A Lab Report On: Implementation on Shortest Distance Algorithm by Dijkstra Method

Submitted By
Name: Dhiraj KC
Roll No. : PUR077BEI014
Faculty: Electronics, Communication and Information
Group: A

Submitted To
Department Of Electronics and Computer Engineering
Checked By: Pukar Karki

TITLE: IMPLEMENTATION ON SHORTEST DISTANCE ALGORITHM BY DIJKSTRA METHOD

THEORY

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a weighted graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later. The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

ALGORITHM

1. Start
2. Mark all nodes unvisited
3. Assign every node a initial value for starting point 0 and for all other ∞
4. For Current Node, compare the lowest weight to go to a neighbour and compare the minimum weight throughout the process
5. Mark the node visited once its updated
6. Follow this process till the node reaches to the end point marking all neighbor and the node as visited
7. Stop

PROGRAM

```
from math import inf
from queue import PriorityQueue

G = {'A': {'B': 3, 'C': 1},
    'B': {'A': 3, 'C': 7, 'D': 5, 'E': 1},
    'C': {'A': 1, 'B': 7, 'D': 2},
    'D': {'C': 2, 'B': 5, 'E': 7},
    'E': {'B': 1, 'D': 7}
    }

def initialize(G, start):
  cost = dict()
  previous = dict()
  for vertex in G:
    cost[vertex] = inf
    previous[vertex] = ''
  cost[start] = 0
  return cost, previous

def relax(G, u, v, cost, previous):
  if (cost[v] > cost[u]+G[u][v]):
    cost[v] = cost[u] + G[u][v]
    previous[v] = u
  return cost, previous
```

```
def DJ(G, start):
  cost, previous = initialize(G, start)
  visited = list()
  Q = PriorityQueue()
  for vertex in G:
    Q.put((cost[vertex], vertex))
  while (Q.empty() == False):
    c, u = Q.get()
    visited.append(u)
    for chimike in G[u]:
      if chimike not in visited:
        cost, previous = relax(G, u, chimike, cost, previous)
  return cost, previous

def constructPath(previous, start, end):
  path = end
  while (previous[end] != ''):
    path = previous[end]+'->'+path
    end = previous[end]
  return path

start = 'A'
cost, previous = DJ(G, start)
for vertex in G:
  print("Path starts from ", start, " to ", vertex, " is ", constructPath(previous, start, vertex))
```

OUTPUT

```
Path starts from  A  to  A  is  A
Path starts from  A  to  B  is  A->B
Path starts from  A  to  C  is  A->C
Path starts from  A  to  D  is  A->C->D
Path starts from  A  to  E  is  A->B->E
```