Name :- Kaustubh Shrikant Kabra

Class:- TE Computer

ERP :-38

Subject :-LP2(AI) (Single Source Shortest Path)

Code:-

```
import sys
from heapq import heappop, heappush


# A class to store a heap node
class Node:
    def __init__(self, vertex, weight=0):
        self.vertex = vertex
        self.weight = weight

    # Override the __lt__() function to make `Node` class work with a min-heap
    def __lt__(self, other):
        return self.weight < other.weight


# A class to represent a graph object
class Graph:
    def __init__(self, edges, n):
        # allocate memory for the adjacency list
        self.adjList = [[] for _ in range(n)]

        # add edges to the directed graph
        for (source, dest, weight) in edges:
            self.adjList[source].append((dest, weight))


def get_route(prev, i, route):
    if i >= 0:
        get_route(prev, prev[i], route)
        route.append(i)


# Run Sinlge Source Shortest Path's algorithm on a given graph
def findShortestPaths(graph, source, n):
    # create a min-heap and push source node having distance 0
    pq = []
    heappush(pq, Node(source))

    # set initial distance from the source to `v` as infinity
    dist = [sys.maxsize] * n
```

```
    # distance from the source to itself is zero
    dist[source] = 0

    # list to track vertices for which minimum cost is already found
    done = [False] * n
    done[source] = True

    # stores predecessor of a vertex (to a print path)
    prev = [-1] * n

    # run till min-heap is empty
    while pq:

        node = heappop(pq)  # Remove and return the best vertex
        u = node.vertex  # get the vertex number

        # do for each neighbor `v` of `u`
        for (v, weight) in graph.adjList[u]:
            if not done[v] and (dist[u] + weight) < dist[v]:  # Relaxation step
                dist[v] = dist[u] + weight
                prev[v] = u
                heappush(pq, Node(v, dist[v]))

        # mark vertex `u` as done so it will not get picked up again
        done[u] = True

    route = []
    for i in range(n):
        if i != source and dist[i] != sys.maxsize:
            get_route(prev, i, route)
            print(f'Path ({source} —> {i}): Minimum cost = {dist[i]}, Route = {route}')
            route.clear()


if __name__ == '__main__':

    # initialize edges as per the above diagram
    # (u, v, w) represent edge from vertex `u` to vertex `v` having weight `w`
    edges = [(0, 1, 10), (0, 4, 3), (1, 2, 2), (1, 4, 4), (2, 3, 9), (3, 2, 7),
             (4, 1, 1), (4, 2, 8), (4, 3, 2)]

    # total number of nodes in the graph (labelled from 0 to 4)
    n = 5

    # construct graph
    graph = Graph(edges, n)
```

```
# run the Sinlge Source Shortest Path's algorithm from every node
for source in range(n):
    findShortestPaths(graph, source, n))
```

Output:-

```
C:\Users\asus\PycharmProjects\LP2(codes)\Scripts\python.exe "C:/Users/asus/PycharmProjects/LP2(codes)/Single Spurce Shortest Path.py"
Path (0 -> 1): Minimum cost = 4, Route = [0, 4, 1]
Path (0 -> 2): Minimum cost = 6, Route = [0, 4, 1, 2]
Path (0 -> 3): Minimum cost = 5, Route = [0, 4, 3]
Path (0 -> 4): Minimum cost = 3, Route = [0, 4]
Path (1 -> 2): Minimum cost = 2, Route = [1, 2]
Path (1 -> 3): Minimum cost = 6, Route = [1, 4, 3]
Path (1 -> 4): Minimum cost = 4, Route = [1, 4]
Path (2 -> 3): Minimum cost = 9, Route = [2, 3]
Path (3 -> 2): Minimum cost = 7, Route = [3, 2]
Path (4 -> 1): Minimum cost = 1, Route = [4, 1]
Path (4 -> 2): Minimum cost = 3, Route = [4, 1, 2]
Path (4 -> 3): Minimum cost = 2, Route = [4, 3]

Process finished with exit code 0
```