Program:

import heapq

def dijkstra(adj\_matrix, start):

n = len(adj\_matrix)

distances = [float('inf')] \* n

distances[start] = 0

visited = [False] \* n

previous = [None] \* n

priority\_queue = [(0, start)] # (distance, node)

while priority\_queue:

current\_dist, current\_node = heapq.heappop(priority\_queue)

if visited[current\_node]:

continue

visited[current\_node] = True

for neighbor in range(n):

weight = adj\_matrix[current\_node][neighbor]

if weight > 0 and not visited[neighbor]:

distance = current\_dist + weight

if distance < distances[neighbor]:

distances[neighbor] = distance

previous[neighbor] = current\_node

heapq.heappush(priority\_queue, (distance, neighbor))

return distances, previous

# Example usage:

adj\_matrix = [

[0, 6, 0, 1, 0],

[6, 0, 5, 2, 2],

[0, 5, 0, 0, 5],

[1, 2, 0, 0, 1],

[0, 2, 5, 1, 0]

]

start\_node = 0 # Starting point x

distances, previous = dijkstra(adj\_matrix, start\_node)

print("Shortest distances from node", start\_node, ":", distances)

output:

Shortest distances from node 0 : [0, 3, 7, 1, 2]