EXPERIMENT – 8

Student Name: Gaganjot Singh **UID: 22BCS14843**
Branch: BE-CSE **Section/Group:** 22BCS_JT_802-B
Semester: 5th **Date of Performance:** 2024
Subject Name: Design & Analysis of Algorithms **Subject Code:** 22CSH-311

1. **Aim:** Develop a program and analyze complexity to find the shortest paths in a graph with positive edge-weights using Dijkstra's algorithm.

2. **Objective:** The objective of this program is to implement Dijkstra's algorithm to find the shortest paths from a source vertex to all other vertices in a graph with positive edge weights. By using an adjacency matrix to represent the graph and an iterative approach to update the shortest distances, the program ensures that the shortest path from the source to each vertex is determined efficiently. The algorithm is crucial for applications such as network routing, navigation systems, and optimizing paths in weighted graphs, where determining minimal travel costs or distances is essential.

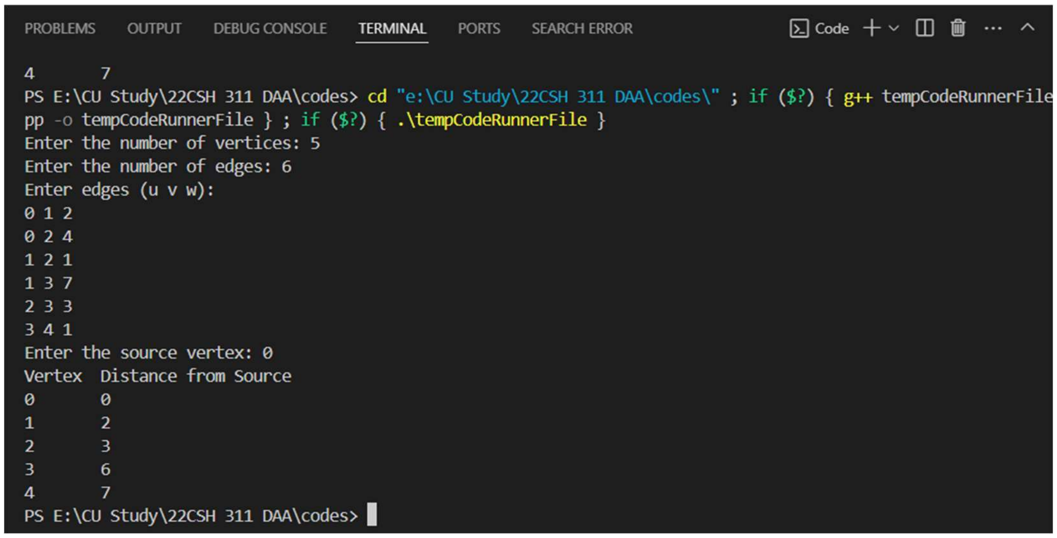
3. Algorithm

- Create a distance array dist[] and set all values to infinity (or a large value), except for the source vertex, where dist[src] = 0.
- Create a boolean array sptSet[] (shortest path tree set) to track which vertices have been processed. Initialize all entries as false.
- Repeat for V−1 vertices (where Vis the number of vertices):
- Pick the vertex u with the minimum distance from the set of vertices not yet processed. This vertex is the one with the smallest value in dist[] that is not in sptSet[].
- Mark vertex u as processed: sptSet[u] = true.
- Update the distance values of all adjacent vertices of u:
- For each vertex v adjacent to u, check if v is not processed, and if there is a path from u to v.
- If the total distance from the source to v through u is smaller than the current value of dist[v], update dist[v].
- End of Loop: After V−1V-1V−1 iterations, the distance array dist[] will contain the shortest distances from the source vertex to all other vertices.
- Print or return the dist[] array, showing the shortest distances from the source to each vertex

4. Implementation/Code:

```
#include <iostream>
#include <climits>
using namespace std;
const int MAX = 100;
struct Edge
{int v;
 int weight;};

int minDistance(int dist[], bool sptSet[], int V)
{int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++)
   {if (!sptSet[v] && dist[v] <= min)
      {min = dist[v], min_index = v;}}
   return min_index;
}

void dijkstra(int graph[MAX][MAX], int V, int src)
{int dist[MAX];
 bool sptSet[MAX];
   for (int i = 0; i < V; i++)
   {dist[i] = INT_MAX;
      sptSet[i] = false;}

   dist[src] = 0;

   for (int count = 0; count < V - 1; count++)
   {int u = minDistance(dist, sptSet, V);
      sptSet[u] = true;
      for (int v = 0; v < V; v++)
      {
         if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v])
         { dist[v] = dist[u] + graph[u][v];
         }}}

   cout << "Vertex\tDistance from Source\n";
   for (int i = 0; i < V; i++)
   { cout << i << " \t" << dist[i] << endl;}}

int main()
{
   int V, E, u, v, w, src;
   cout << "Enter the number of vertices: ";
   cin >> V;
   cout << "Enter the number of edges: ";
   cin >> E;
   int graph[MAX][MAX] = {0};
   cout << "Enter edges (u v w):\n";
   for (int i = 0; i < E; i++)
   {cin >> u >> v >> w;
    graph[u][v] = w;}

   cout << "Enter the source vertex: ";
   cin >> src;
   dijkstra(graph, V, src);
   return 0;}
```

5. Output:



```
4       7
PS E:\CU Study\22CSH 311 DAA\codes> cd "e:\CU Study\22CSH 311 DAA\codes\" ; if ($?) { g++ tempCodeRunnerFile
pp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter the number of vertices: 5
Enter the number of edges: 6
Enter edges (u v w):
0 1 2
0 2 4
1 2 1
1 3 7
2 3 3
3 4 1
Enter the source vertex: 0
Vertex  Distance from Source
0       0
1       2
2       3
3       6
4       7
PS E:\CU Study\22CSH 311 DAA\codes>
```

6. Time Complexity :

1. Finding the minimum distance vertex in the minDistance function takes O(V) time for each vertex, leading to O(V^2) overall for this step.
2. Updating the distance of adjacent vertices takes O(1) for each edge, resulting in a total of O(E), where EEE is the number of edges.

Therefore, the overall time complexity is O(V^2), which is higher than the priority queue implementation but is straightforward without requiring additional data structures like vector or priority_queue.

7. Learning Outcomes:

- Understood how to represent graphs using adjacency matrices and lists.
- Gained knowledge on how Dijkstra's algorithm finds the shortest paths from a source vertex to all other vertices in a graph with positive edge weights.
- Understood the greedy approach used in Dijkstra’s algorithm to progressively find the minimum distance vertex.
- Implemented updating and maintain the shortest known distances to adjacent vertices.
- Develop an understanding of the time complexity and performance trade-offs in implementing graph algorithms.