Experiment 9

Student Name: SANSKAR AGRAWAL UID: 20BCS5914

Branch: CSE Section/Group: 806-B

Semester: 5 Subject Code: 20CSP-312

Subject Name: Data Analysis and Algorithm

Aim:

To create a code to find the shortest path in a graph using Dijkstra's algorithm.

Algorithm/Pseudocode:

Working of Dijkstra's Algorithm:

Step 1) Initialize the starting node with 0 costs and the rest of the node as Infinity Cost.

Step 2) Maintain an array or list to keep track of the visited nodes

Step 3) Update the node cost with the minimum cost. It can be done by comparing the current cost with the path cost.

Step 4) Continue step 3 until all the node is visited.

After completing all these steps, we will find the path that costs a minimum from source to destination.

Pseudocode for Dijkstra's Shortest Path:

```
Dijkstra(G, S):
  for each vertex V in G
distance[V] & lt; - Infinity
previous[V] & lt; - NULL if V
does not equal S, then, (priority
queue) Q.push(V) distance[S]
= 0
  While Q is not empty
   U & lt; - Extract the MIN from Q
   For each unvisited adjacent V of U    TotalDistance
& lt; - distance[U] + edge_cost(U, V)  if
TotalDistance is less than distance[V], then
distance[V] & lt; - TotalDistance  previous[V] & lt; -
u return distance, previous
```

☐ Code:

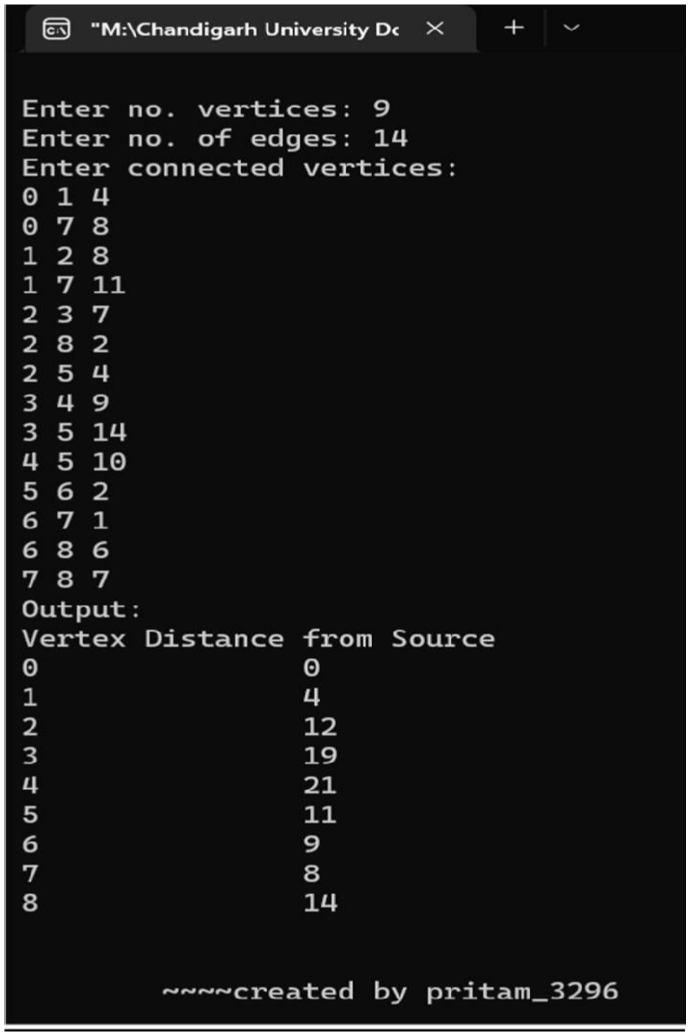
```
//find Dijkstra's shortest path using priority_queue in STL #include
<bits/stdc++.h> using namespace std;
#define INF 0x3f3f3f3f

typedef pair<int, int> iPair;

class Graph {     int V;     list<pair<int,
int> >* adj;
```

```
public:     Graph(int V);     void
addEdge(int u, int v, int w); void shortestPath(int
s);
};

Graph::Graph(int V) { this-
>V = V; adj = new
list<iPair>[V];
}

void Graph::addEdge(int u, int v, int w)
{ adj[u].push_back(make_pair(v, w));
    adj[v].push_back(make_pair(u, w));
}

void Graph::shortestPath(int src)
{ priority_queue<iPair, vector<iPair>, greater<iPair> >pq;



   vector<int> dist(V, INF);
pq.push(make_pair(0, src)); dist[src]
= 0;          while (!pq.empty())
                                                                                                    {
          int u = pq.top().second;
pq.pop();  list<pair<int,  int>  >::iterator  i;  for  (i  =
adj[u].begin(); i != adj[u].end(); ++i)
          {    int v = (*i).first;          int weight =
(*i).second;          if
(dist[v] > dist[u] + weight)
                {
                   dist[v] = dist[u] + weight;
pq.push(make_pair(dist[v], v));
             }
          }
        }
    printf("Vertex          Distance          from
Source\n");     for (int i = 0; i < V; ++i)          cout<<i<<"
\t\t"<<dist[i]<<endl;
}

int main() {      cout<<"Find Dijkstra's Shortest
Path: \n"<<endl; int V,n; cout<<"Enter
    no. vertices: "; cin>>V;
cout<<"Enter no. of edges: "; cin>>n;
    Graph g(V);


    cout<<"Enter        connected        vertices:
"<<endl;     int x,y,z;     for(int i=0;i<n;i++)
    {
        cin>>x>>y>>z;
        g.addEdge(x,y,z);
    }


cout<<"Output:"<<endl;
    g.shortestPath(0);

    cout<<"\n\n\t~~~~created by pritam_3296"<<endl;
return 0;
}
```

Complexity Analysis:

Time complexity is O(E * logV), Where E is the number of edges and V is the number of vertices.. Space complexity is O(V).

☐ Output:



"M:\Chandigarh University Dc

```
Enter no. vertices: 9
Enter no. of edges: 14
Enter connected vertices:
0 1 4
0 7 8
1 2 8
1 7 11
2 3 7
2 8 2
2 5 4
3 4 9
3 5 14
4 5 10
5 6 2
6 7 1
6 8 6
7 8 7
Output:
Vertex Distance from Source
0               0
1               4
2               12
3               19
4               21
5               11
6               9
7               8
8               14


         ~~~~created by pritam_3296
```

☐ Learning Outcome:

- We learnt about Dijkstra's algorithm and its implementation.
- We learnt about shortest path in a graph.
- We learnt about practical application of Dijkstra's algorithm in real life problems.
- We learnt to calculate time complexity of programs and thereby create the most optimal program possible.

Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			

2.			
3.			