EXPERIMENT NO.6

Single Source Shortest Path Using Dijkstra's Algorithm

Reg No.:-24141045

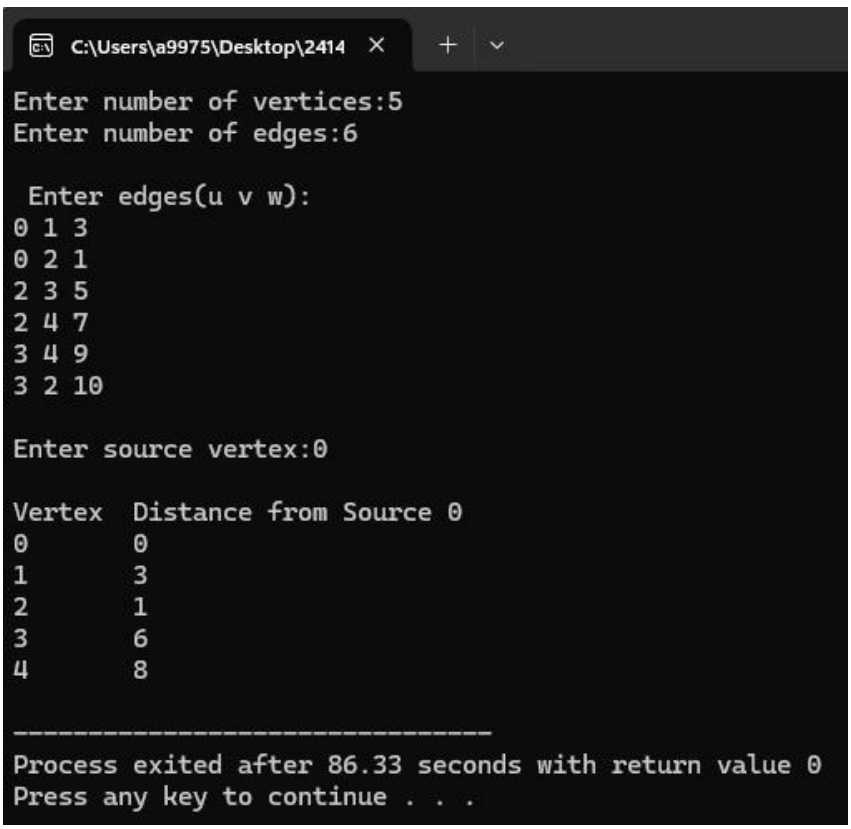
Program:-

```
 #include <bits/stdc++.h>
Using namespace std;
Struct Edge{
Int vertex;
Int weight;
};
Void dijkstra(int V, vector< vector < Edge > > &adj, int src) {
   Vector<int> dist(V, INT_MAX);
   Dist[src] = 0;
   Int i;
   Priority_queue<pair <int,int>, vector<pair <int,int> >, greater<pair <int,int> >
> pq;
   Pq.push({0, src});
   While (!pq.empty()) {
      Int d = pq.top().first;
      Int u = pq.top().second;
      Pq.pop();
```

```
      If (d > dist[u]) continue;
      For (i=0; i<adj[u].size(); i++) {
        Int v = adj[u][i].vertex;
        Int w = adj[u][i].weight;
        If (dist[u] + w < dist[v]) {
          Dist[v] = dist[u] + w;
          Pq.push({dist[v], v});
        }
      }
    }
    Cout << "\nVertex\tDistance from Source " << src << "\n";
    For (i = 0; i < V; ++i)
      Cout << i << "\t" << dist[i] << "\n";
}
Int main() {
    Int i, V , E;
    Cout<<"Enter number of vertices:";
    Cin>>V;
    Cout<<"Enter number of edges:";
    Cin>>E;
    Vector< vector< Edge > > adj(V);
    Cout<<"\n Enter edges(u v w):\n";
    For(i=0; i<E; i++){
```

```
    Int u,v,w;
    Cin>>u>>v>>w;
    Adj[u].push_back({v,w});
    Adj[v].push_back({u,w});
}
    Int src;
    Cout<<"\nEnter source vertex:";
    Cin>> src;
    Dijkstra(V, adj, src);
    Return 0;
}
```

Output:-



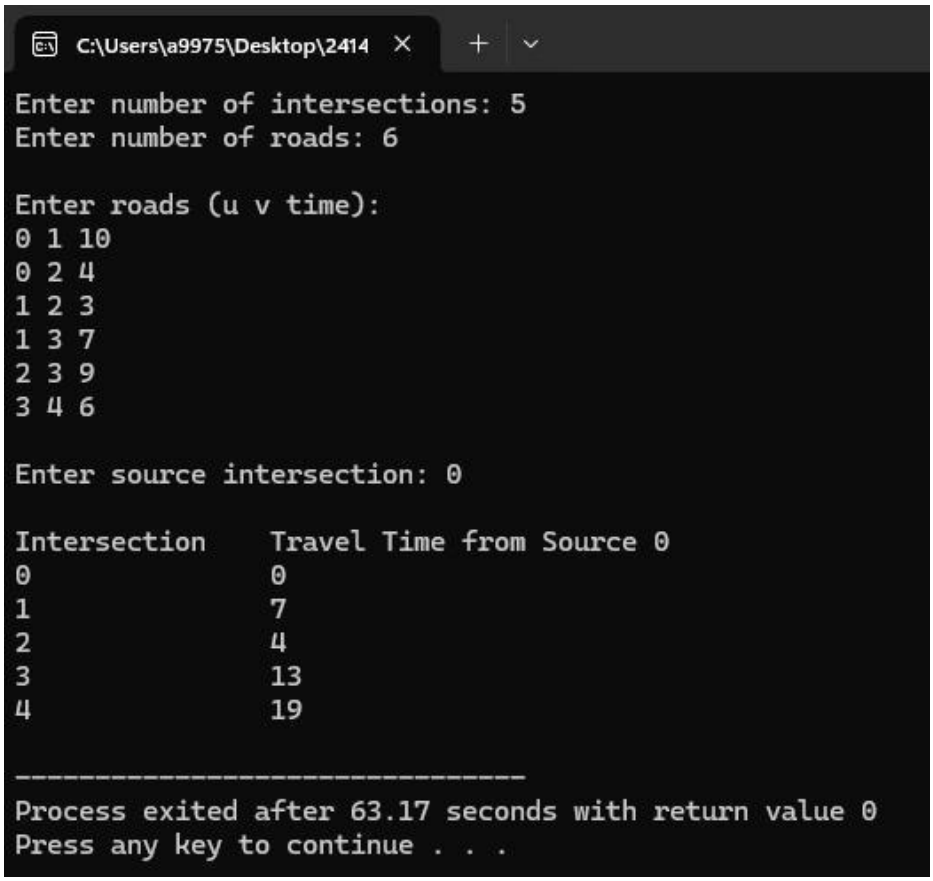
Application:-

```
#include <bits/stdc++.h>
Using namespace std;
Struct Edge {
   Int vertex;
   Int weight;
};
Void dijkstra(int V, vector< vector < Edge > > &adj, int src) {
   Vector<int> dist(V, INT_MAX);
   Dist[src] = 0;
   Int i;
   Priority_queue<pair <int, int>, vector< pair <int, int> >, greater< pair <int, int
> > > pq;
   Pq.push({0, src});
   While (!pq.empty()) {
      Int d = pq.top().first;
      Int u = pq.top().second;
      Pq.pop();
      If (d > dist[u]) continue;
      For (i = 0; i < adj[u].size(); i++) {
         Int v = adj[u][i].vertex;
         Int w = adj[u][i].weight;
         If (dist[u] + w < dist[v]) {
```

```
            Dist[v] = dist[u] + w;
            Pq.push({dist[v], v});
          }
        }
    }
    Cout << “\nIntersection\tTravel Time from Source “ << src << “\n”;
    For (i = 0; i < V; i++) {
      If (dist[i] == INT_MAX)
        Cout << i << “\t\tINF\n”;
      Else
        Cout << i << “\t\t” << dist[i] << “\n”;
    }
}
Int main() {
    Int V, E;
    Cout << “Enter number of intersections: “;
    Cin >> V;
    Cout << “Enter number of roads: “;
    Cin >> E;
    Vector<vector < Edge > > adj(V);
    Cout << “\nEnter roads (u v time):\n”;
    For (int i = 0; i < E; i++) {
      Int u, v, time;
```

```
    Cin >> u >> v >> time;
    Adj[u].push_back({v, time});
    Adj[v].push_back({u, time}); // undirected
  }
  Int src;
  Cout << "\nEnter source intersection: ";
  Cin >> src;
  Dijkstra(V, adj, src);
  Return 0;
}
```

Output:-



Algorithm Steps:

1. Initialize distances from the source to all vertices as ∞, and distance to source = 0.
2. Mark all vertices as unvisited.
3. Choose the unvisited vertex with the smallest known distance.
4. For each neighbor of this vertex:

Calculate new_distance = current_distance + edge_weight.

If new_distance < old_distance, update it.

5. Mark the vertex as visited (it's now finalized).
6. Repeat until all vertices are visited or shortest paths are found.

Time Complexity:

Using adjacency matrix → $O(V^2)$

Using min-heap (priority queue) → O(E log V)

List of Applications:-

1. GPS and navigation systems (shortest route finding)
2. Network routing (like in OSPF protocol)
3. Airline flight path optimization
4. Robot motion planning
5. Social network analysis (finding degrees of separation)
6. Urban traffic management systems
7. Game development (AI pathfinding)
8. Internet packet switching and data transmission
9. Supply chain and logistics optimization