Khulna University of Engineering & Technology

KUET

SESSIONAL REPORT

Course No. : CSE 2202

Department Of Computer Science and Engineering

Experiment No. : 01

Name of the Experiment : Implementation and Analysis of Dijkstra's shortest path algorithm.

Remarks

Name : Dadhichi Sarker Shayon

Roll No. : 2207118

Group No. : B2

Date of Performance : 06/10/2025

Year : 2nd

Date of Submission : 13/10/2025

Term : 2nd

Title:

Implementation and analysis of Dijkstra’s shortest path algorithm .

Objective:

1.Implementing Dijkstra’s algorithm to find the shortest paths from a given source vertex to all other vertices in a weighted graph with non-negative edge weights..
2.Analyzing the time complexity and performance of Dijkstra’s algorithm.
3.Analyzing the limitations of Dijkstra’s algorithm.

Theory:

Dijkstra’s algorithm was proposed by Edsger W. Dijkstra in 1956.It is a greedy algorithm used to find the shortest path between nodes in a graph. It is applicable to graphs with non-negative edge weights and can be used to determine the shortest distance from a single source to all vertices.Dijkstra’s algorithm works only for non-negative edge weights.It produces the correct shortest path tree.

Working Principle:

Initializing all vertex distances as infinity except for the source vertex which has distance zero.Using a min-priority queue or set to always pick the vertex u with the smallest distance value.For each neighbor v of u, if the path dist[u] + weight(u, v) is smaller than dist[v], updating it.Repeating until all vertices have been processed or the priority queue or set becomes empty.This ensures that each vertex is relaxed at most once with its final shortest distance.

Pseudocode:

```
 Initialize array dist[] of size |V| with infinity
Set dist[source]  to  0

Create a min-priority queue pq
Insert (0, source) into pq

While pq is not empty:
    (d, u) =pq.extractMin()

    If d > dist[u]:
       Continue
    For each neighbor (v, w) of u in adjacency list:
      If dist[u] + w < dist[v]:
         dist[v] = dist[u] + w
         pq.insert( (dist[v], v) )
         Return dist[]
```

Implementation:

```
#include<bits/stdc++.h>
using namespace std;

void add_edge(int u, int v, int w, vector<vector<pair<int,int>>>& adj)
{
    adj[u].push_back({v,w});
    //adj[v].push_back({u,w});
}
void dijkstra(int src, vector<vector<pair<int,int>>>& adj, vector<int>& dist)
{
    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
    pq.push({0, src});
    dist[src] = 0;
  while(!pq.empty())
    {
        int d = pq.top().first;
        int u = pq.top().second;
        pq.pop();
        if (d != dist[u]) continue;
              for (auto it : adj[u])
        {
            int v = it.first;
            int w = it.second;
            if (dist[u] + w < dist[v])
            {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}

int main()
{
    int n, m;
    cin >> n >> m;
   vector<vector<pair<int,int>>> adj(n);
    for (int i=0; i<m; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        add_edge(u, v, w, adj);
     }

   vector<int> dist(n, INT_MAX);
    int src;
    cin >> src;
    dijkstra(src, adj, dist);
  for (int i=0; i<n; i++)
    {
        if (dist[i] == INT_MAX)
            cout << "Distance of node " << i << " from source " << src << " is INF\n";
        else
            cout << "Distance of node " << i << " from source " << src << " is " << dist[i] <<
"\n";
    }}
```

Sample input & output:

```
5 6
0 1 2
0 2 4
1 2 1
1 3 7
2 4 3
3 4 1
0
```

```
Distance of node 0 from source 0 is 0
Distance of node 1 from source 0 is 2
Distance of node 2 from source 0 is 3
Distance of node 3 from source 0 is 9
Distance of node 4 from source 0 is 6
```

Time Complexity:

Using min-heap or priority queue gives a time complexity of O((V+E)logV)and space Complexity of O(V+E).

Discussion:

Dijkstra's Algorithm is a greedy shortest-path algorithm that systematically explores a weighted graph to compute the shortest distance from a given source node to every other node. The fundamental idea is to always choose the next vertex with the smallest distance.The algorithm maintains a priority queue to efficiently get the vertex with the smallest current distance. Each time a vertex u is processed, its neighbors v are relaxed , meaning if the path through u provides a shorter route to v, the distance to v is updated.The condition if (d != dist[u]) continue; ensures that outdated entries are skipped when a better path has already been discovered, preventing redundant work and infinite loops. The adjacency list representation of the graph ensures that every edge is explored exactly once, making the algorithm efficient for sparse graphs.Dijkstra's algorithm is widely used in network routing, GPS navigation, and real-time systems. However, it assumes non-negative edge weights if negative weights are present, Bellman-Ford must be used instead.The efficiency depends heavily on the data structures used .Using a binary heap or priority queue, the complexity is O((V+E)logV), which works well for sparse graphs.Using an adjacency matrix, it becomes O(V^2), suitable for dense graphs or small datasets.In practical applications, Dijkstra's Algorithm demonstrates an excellent trade-off between speed and accuracy. It guarantees optimal results in graphs where all edges have non-negative weights and can easily be extended to retrieve actual shortest paths by maintaining a parent vector.

Conclusion:

Dijkstra’s Algorithm is one of the most fundamental and efficient techniques for solving the single source shortest path problem on weighted graphs with non negative edges. It efficiently computes the shortest distances using a greedy strategy and a priority queue. The algorithm’s performance and accuracy make it a cornerstone of modern graph based problem solving, particularly in navigation systems, computer networks.

References:

1. Lab slides.
2. Dijkstra’s Algorithm for Adjacency List Representation | Greedy Algo-8 – Geeks for Geeks
3. Dijkstra's Shortest Path Algorithm using priority_queue of STL - Geeks for Geeks