Name: Achyut Shukla Batch: AA – A1 PRN: 20070122005

Dijkstra’s Algorithm

Dijkstra's algorithm is a popular algorithm used to find the shortest path in a weighted graph from a single source vertex to all other vertices. It was developed by Edsger W. Dijkstra in 1956 and is widely used in various applications, such as routing in computer networks, finding shortest routes in transportation systems, and solving optimization problems.

The algorithm maintains a set of unvisited vertices and a set of visited vertices. It starts at the source vertex and iteratively selects the vertex with the smallest known distance from the source that has not yet been visited. It then explores all its neighboring vertices, updating their distances from the source if a shorter path is found. This process continues until all vertices have been visited or the target vertex (if specified) is reached.

Code

```
 #include <bits/stdc++.h>

using namespace std;


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<pair<int, int>>[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<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>
        pq;

    vector<int> dist(V, INT_MAX);

    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)
        printf("%d \t\t %d\n", i, dist[i]);
}

int main()
{
    int V = 9;
    Graph g(V);

    g.addEdge(0, 1, 4);
    g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);
    g.addEdge(2, 3, 7);
    g.addEdge(2, 8, 2);
    g.addEdge(2, 5, 4);
    g.addEdge(3, 4, 9);
    g.addEdge(3, 5, 14);
    g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 1);
    g.addEdge(6, 8, 6);
    g.addEdge(7, 8, 7);

    g.shortestPath(5);

    return 0;
}
```

Name: Achyut Shukla Batch: AA – A1 PRN: 20070122005

Complexity analysis of Dijkstra's Algorithm

Time Complexity - O(|V| + |E| log |V|)

Dijkstra's algorithm has several advantages and disadvantages, which are outlined below:

Advantages of Dijkstra's Algorithm:

Finds the shortest path: Dijkstra's algorithm guarantees finding the shortest path in terms of the sum of edge weights, from a single source vertex to all other vertices in a weighted graph, as long as all edge weights are non-negative. It provides an optimal solution to the shortest path problem.

Efficient for small graphs with non-negative edge weights: Dijkstra's algorithm can be efficient for small graphs with non-negative edge weights, especially when implemented with a priority queue (e.g., min-heap), which allows for efficient selection of the vertex with the smallest distance in each iteration.

Versatile: Dijkstra's algorithm can be used in various applications, such as routing in computer networks, finding shortest routes in transportation systems, and solving optimization problems. It is a widely used algorithm with many real-world applications.

Disadvantages of Dijkstra's Algorithm:

Inefficient for large graphs: Dijkstra's algorithm with a naive implementation using an adjacency matrix has a time complexity of O(|V|^2), which can be inefficient for large graphs with many vertices. The time complexity can be improved to O(|V| + |E| log |V|) with a priority queue, but it may still be slow for very large graphs.

Doesn't handle negative edge weights: Dijkstra's algorithm assumes that all edge weights are non-negative. If there are negative edge weights in the graph, the algorithm may not work correctly, as it does not handle negative weight cycles. In such cases, other algorithms such as Bellman-Ford or Floyd-Warshall may be more appropriate.

Limited to single source shortest path: Dijkstra's algorithm finds the shortest path from a single source vertex to all other vertices in the graph. If multiple source vertices or multiple pairs of source and target vertices need to be considered, multiple invocations of the algorithm may be required, which can be inefficient.