



Experiment 3.2

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

Objectives: Code and analyze to find shortest paths in a graph with positive edge weights using Dijkstra's.

Input/Apparatus Used: Graph ($G = (V, E)$) is taken as input for this problem.

Procedure/Algorithm:

- Create a set sptSet (shortest path tree set) that keeps track of vertices included in the shortest-path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
- Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign the distance value as 0 for the source vertex so that it is picked first. While sptSet doesn't include all vertices
- Pick a vertex u which is not there in sptSet and has a minimum distance value.
- Include u to sptSet.
- Then update distance value of all adjacent vertices of u .
- To update the distance values, iterate through all adjacent vertices.
- For every adjacent vertex v , if the sum of the distance value of u (from source) and weight of edge $u-v$, is less than the distance value of v , then update the distance value of v .

Sample Code:

```
import java.util.*;

public class DijkstraAlgo {
    static class Edge {
        int src, dest, wt;
        public Edge(int s, int d, int w) {
            this.src = s;
            this.dest = d;
            this.wt = w;
        }
    }
    private static void createGraph(ArrayList<Edge> graph[]) {
        for (int i = 0; i < graph.length; i++) {
            graph[i] = new ArrayList<>();
        }
    }
}
```



```
graph[0].add(new Edge(0, 1, 2));
graph[0].add(new Edge(0, 2, 4));

graph[1].add(new Edge(1, 3, 7));
graph[1].add(new Edge(1, 2, 1));

graph[2].add(new Edge(2, 4, 3));

graph[3].add(new Edge(3, 5, 1));

graph[4].add(new Edge(4, 3, 2));
graph[4].add(new Edge(4, 5, 5));

}

static class Pair implements Comparable<Pair>{
    int n, path;
    public Pair(int n, int path){
        this.n = n;
        this.path = path;
    }

    @Override
    public int compareTo(Pair p2){
        return this.path - p2.path; // path based sorting for my pairs
    }
}

public static void dijkstra(ArrayList<Edge> graph[], int src){
    int dist[] = new int[graph.length];
    for(int i=0; i<graph.length; i++){
        if( i != src){
            dist[i] = Integer.MAX_VALUE; // +infinity
        }
    }
    boolean vis[] = new boolean[graph.length];
    PriorityQueue<Pair> pq = new PriorityQueue<>();
    pq.add(new Pair(src, 0));
    //loop
    while(!pq.isEmpty()){
        Pair curr = pq.remove();
        if(!vis[curr.n]){
            vis[curr.n] = true;
            //neighbours
            for(int i=0; i<graph[curr.n].size(); i++){
                Edge e = graph[curr.n].get(i);
                int u = e.src;
```



Course Name: DAA Lab

Course Code: 21ITH-311/21CSH-311

```
int v= e.dest;
int wt = e.wt;

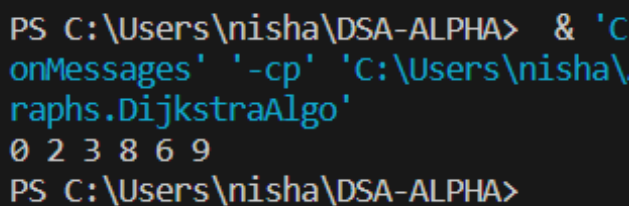
if(dist[u] + wt < dist[v]){
    dist[v] = dist[u] + wt;
    pq.add(new Pair(v, dist[v]));
}
}
}

//print all source to vertices shortest dist
for(int i=0; i<dist.length; i++){
    System.out.print(dist[i] + " ");
}
System.out.println();
}

public static void main(String[] args) {
    int V = 6;
    ArrayList<Edge> graph[] = new ArrayList[V];
    createGraph(graph);

    int src = 0;
    dijkstra(graph, src);
}
}
```

Observations/Outcome :



```
PS C:\Users\nisha\DSA-ALPHA> java -cp 'C:\Users\nisha\DSA-ALPHA\bin' DijkstraAlgo
0 2 3 8 6 9
PS C:\Users\nisha\DSA-ALPHA>
```

Time Complexity: $O(V + E)$