Assignment 10

Sayak Ghorai || BT21GCS004 || B2 || Design and Analysis of Algorithm

Q: Write a code to demonstrate Dijkestra's Algorithm to find shortest path from source node to any other node.

Answer:

Approach: first make a graph with certain Edges and Vertecies and then add edges between Vertecies. That creates a Graph Structure. Now we can easily apply Dijkestra's Algo in this structure.

Code:

DijkstraAlgorithm.java

```java
public class DijkstraAlgorithm {
    1 usage
    private static int getMinDistance(int[] DiatanceArr, boolean[] visited) {
        int minDistance = Integer.MAX_VALUE;
        int minIndex=-1;
        for (int i = 0; i < DiatanceArr.length; i++) {
            if (!visited[i] && DiatanceArr[i] < minDistance) {
                minDistance = DiatanceArr[i];
                minIndex = i;
            }
        }
        return minIndex;
    }
    1 usage
    public static void dijkstra(int[][] graph, int source) {
        int numVertices = graph.length;
        int[] DistanceArr = new int[numVertices];
        boolean[] visited = new boolean[numVertices];

        for (int i = 0; i < numVertices; i++) {
            DistanceArr[i] = Integer.MAX_VALUE;
        }
        DistanceArr[source] = 0;
        for (int i = 0; i < numVertices - 1; i++) {
            int u = getMinDistance(DistanceArr, visited);
            visited[u] = true;
            for (int v = 0; v < numVertices; v++) {
                if(!visited[v] && graph[u][v] != 0 && DistanceArr[u] != Integer.MAX_VALUE && DistanceArr[u] + graph[u][v] < DistanceArr[v]){
                    DistanceArr[v] = DistanceArr[u] + graph[u][v];
                }
            }
        }

        for (int i = 0; i < numVertices; i++) {
            System.out.println("Distance from " + source + " to " + i + " is »»»» " + DistanceArr[i]);
        }
    }

    public static void main(String[] args) {
        int[][] graph = {
                {0, 2, 5, 0, 2},
                {2, 0, 2, 5, 0},
                {5, 2, 0, 1, 3},
                {0, 5, 1, 0, 2},
                {2, 0, 3, 2, 0}
        };
        int source = 0;
        dijkstra(graph, source);
    }
}
```

Output:

```
/Users/sayakghorai/Desktop/DAA_Assignments/Assignment_10/out/production/Assignment_10 DijkstraAlgorithm
Distance from 0 to 0 is »»»» 0
Distance from 0 to 1 is »»»» 2
Distance from 0 to 2 is »»»» 4
Distance from 0 to 3 is »»»» 4
Distance from 0 to 4 is »»»» 2

Process finished with exit code 0
```

Analysis:

Time complexity of Dijkestra's Algorithm is ***O(E+VlogV)****(using priority queue)* or ***O(ElogV)****(using max-heap)* or ***O(V^2)***, where Bellman Ford is having O(EV) time complexity.

Dijkestra's algorithm is more efficient but it can't handle Negetive Weighted Edges thats why we use BellmanFord. But for complete graph, BellmanFord is more efficient.