**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: MCA, Semester - I

**Course: Data Structures and Algorithms**

**Faculty:**

LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.09**

**A.1 Aim: Implementation of Greedy Techniques**

**Task 1: Implementation of Greedy Techniques:** **Single source shortest path with its analysis: Dijkstra's algorithm.**

**Task 2: Compete analysis of Dijkstra's algorithm**

****

**A.2 Prerequisite:**

1. Knowledge of adjacency matrix: 2 - D Array Handling

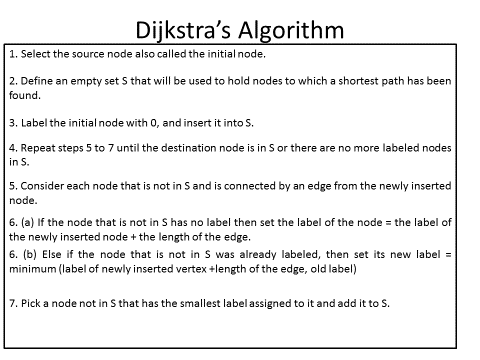
2. Concept of Greedy Method

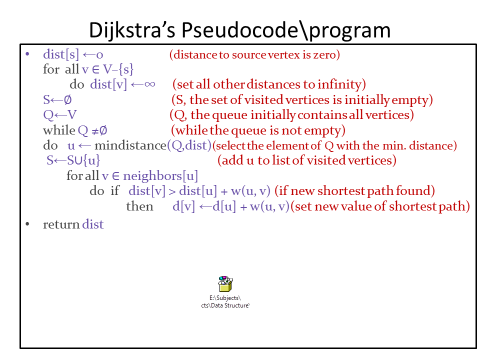
**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Understand the concept of single source shortest path algorithm
2. Write complex program based on graphs
3. Analyze algorithms.

**A.4 Theory:**

****

****

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical.***

***The soft copy must be uploaded on the Portal.)***

|  |  |
| --- | --- |
| Program:MCA | Sem:1 |
| Roll No.A073 | Name:Aryan Srivastava |
| Division: | Batch :3 |
| Date of Experiment: | Date of Submission |
| Grade : |  |

**B.1 Software Code written by student:**

**import java.util.\*;**

**class DijkstraAlgorithm {**

**private static final int NO\_PARENT = -1;**

**public static void dijkstra(int[][] adjacencyMatrix, int startVertex) {**

**int nVertices = adjacencyMatrix[0].length;**

**int[] shortestDistances = new int[nVertices];**

**boolean[] added = new boolean[nVertices];**

**for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++) {**

**shortestDistances[vertexIndex] = Integer.MAX\_VALUE;**

**added[vertexIndex] = false;**

**}**

**shortestDistances[startVertex] = 0;**

**int[] parents = new int[nVertices];**

**parents[startVertex] = NO\_PARENT;**

**for (int i = 1; i < nVertices; i++) {**

**int nearestVertex = -1;**

**int shortestDistance = Integer.MAX\_VALUE;**

**for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++) {**

**if (!added[vertexIndex] && shortestDistances[vertexIndex] < shortestDistance) {**

**nearestVertex = vertexIndex;**

**shortestDistance = shortestDistances[vertexIndex];**

**}**

**}**

**added[nearestVertex] = true;**

**for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++) {**

**int edgeDistance = adjacencyMatrix[nearestVertex][vertexIndex];**

**if (edgeDistance > 0 && ((shortestDistance + edgeDistance) < shortestDistances[vertexIndex])) {**

**parents[vertexIndex] = nearestVertex;**

**shortestDistances[vertexIndex] = shortestDistance + edgeDistance;**

**}**

**}**

**}**

**printSolution(startVertex, shortestDistances, parents);**

**}**

**private static void printSolution(int startVertex, int[] distances, int[] parents) {**

**int nVertices = distances.length;**

**System.out.println("Vertex\t Distance\tPath");**

**for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++) {**

**if (vertexIndex != startVertex) {**

**System.out.print(startVertex + " -> ");**

**System.out.print(vertexIndex + " \t\t ");**

**System.out.print(distances[vertexIndex] + "\t\t");**

**printPath(vertexIndex, parents);**

**System.out.println();**

**}**

**}**

**}**

**private static void printPath(int currentVertex, int[] parents) {**

**if (currentVertex == NO\_PARENT) {**

**return;**

**}**

**printPath(parents[currentVertex], parents);**

**System.out.print(currentVertex + " ");**

**}**

**public static void main(String[] args) {**

**int[][] adjacencyMatrix = {**

**{0, 5, 0, 0, 0, 10, 0, 0},**

**{5, 0, 3, 0, 0, 0, 0, 0},**

**{0, 3, 0, 8, 0, 0, 0, 0},**

**{0, 0, 8, 0, 2, 0, 7, 0},**

**{0, 0, 0, 2, 0, 1, 0, 6},**

**{10, 0, 0, 0, 1, 0, 4, 0},**

**{0, 0, 0, 7, 0, 4, 0, 2},**

**{0, 0, 0, 0, 6, 0, 2, 0}**

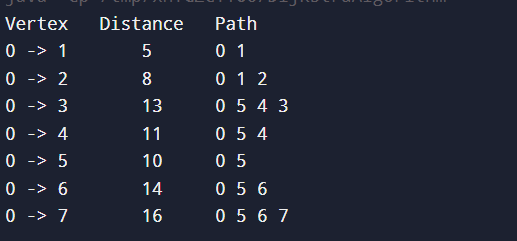
**};**

**dijkstra(adjacencyMatrix, 0);**

**}**

**}**

**B.2 Input and Output:**

****

**B.3 Observations and learning:**

**Finds the shortest path from a single source in weighted graphs by always selecting the nearest unvisited node.**

**B.4 Conclusion:**

**Successfully implemented Dijkstras algorithm**

**B.5 Question of curosity**

***What can be done to avoid the worst case of quick sort***

**To avoid the worst-case in Quick Sort:**

1. **Randomized Pivot: Pick a random element as the pivot to avoid consistently poor splits.**
2. **Median-of-Three Pivoting: Choose the median of the first, middle, and last elements for a balanced partition.**
3. **Hybrid Approach: Switch to Insertion Sort for small subarrays to improve efficiency.**

***Comment on space and time complexity of merge sort***

**Merge Sort Complexity**

* **Time: O(nlog n)O(n \log n)O(nlogn) in all cases.**
* **Space: O(n)O(n)O(n) due to additional arrays needed for merging.**