Ex. No.: 6a GREEDY TECHNIQUE - DIJKSTRA’S ALGORITHM

Date:

AIM:

To write a python program to implement Dijkstra’s algorithm using Greedy technique.

ALGORITHM:

STEP 1: Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

STEP 2: Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

STEP 3: While *sptSet* doesn’t include all vertices:

- Pick a vertex u which is not there in *sptSet* and has minimum distance value.
- Include u to *sptSet*.
- 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 a 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.

PROGRAM:

```
# Python program for Dijkstra's single source shortest path algorithm.

class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)] for row in range(vertices)]

    def printSolution(self, dist):
        print("Vertex \t Distance from Source")
        for node in range(self.V):
            print(node, "\t\t", dist[node])

    def minDistance(self, dist, sptSet):
        min = 1e7
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v
        return min_index
    def dijkstra(self, src):

        dist = [1e7] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):
            u = self.minDistance(dist, sptSet)
            sptSet[u] = True
            for v in range(self.V):
                if (self.graph[u][v] > 0 and
                sptSet[v] == False and
                dist[v] > dist[u] + self.graph[u][v]):
                    dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

# Driver program
g = Graph(9)

g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
           [4, 0, 8, 0, 0, 0, 0, 11, 0],
           [0, 8, 0, 7, 0, 4, 0, 0, 2],
           [0, 0, 7, 0, 9, 14, 0, 0, 0],
           [0, 0, 0, 9, 0, 10, 0, 0, 0],
           [0, 0, 4, 14, 10, 0, 2, 0, 0],
           [0, 0, 0, 0, 0, 2, 0, 1, 6],
           [8, 11, 0, 0, 0, 0, 1, 0, 7],
           [0, 0, 2, 0, 0, 0, 6, 7, 0]
          ]

g.dijkstra(0)
```

OUTPUT:

RESULT:

Thus the python program for Dijkstra’s algorithm using Greedy technique was implemented and executed successfully.

Ex. No.: 6b **HUFFMAN TREES AND CODES**
Date:

AIM:

To write a python program to implement Huffman trees and codes using Greedy technique.

ALGORITHM:

STEP 1: Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)

STEP 2: Extract two nodes with the minimum frequency from the min heap.

STEP 3: Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

STEP 4: Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

PROGRAM:

```
import heapq

class node:
	def __init__(self, freq, symbol, left=None, right=None):
		self.freq = freq
		self.symbol = symbol
		self.left = left
		self.right = right
		self.huff = ''

	def __lt__(self, nxt):
		return self.freq < nxt.freq



def printNodes(node, val=''):
	newVal = val + str(node.huff)
	if(node.left):
		printNodes(node.left, newVal)
	if(node.right):
		printNodes(node.right, newVal)
	if(not node.left and not node.right):
		print(f"{node.symbol} -> {newVal}")

chars = ['a', 'b', 'c', 'd', 'e', 'f']
freq = [5, 9, 12, 13, 16, 45]
nodes = []
```

```
for x in range(len(chars)):
        heapq.heappush(nodes, node(freq[x], chars[x]))

while len(nodes) > 1:
        left = heapq.heappop(nodes)
        right = heapq.heappop(nodes)

        left.huff = 0
        right.huff = 1

        newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

        heapq.heappush(nodes, newNode)

printNodes(nodes[0])
```

OUTPUT:

RESULT:

Thus the python program for the implementation of Huffman Trees and Codes using Greedy technique was done and executed successfully.