COMPUTER NETWORKS LAB

VIVEK RAJEEV

Date __/__/__
Page __
SPLASH

PROGRAM : 7A – DISTANCE VECTOR ALGORITHM

CODE

```
class Topology :

    def __init__(self, array_of_points):
        self.nodes = array_of_points
        self.edges = []

    def add_direct_connection (self, p1, p2, cost):
        self.edges.append((p1, p2, cost))
        self.edges.append((p2, p1, cost))

    def distance_vector_routing (self):
        import collections
        for node in self.nodes :
            dist = collections.defaultdict(int)
            next_hop = {node : node}
            for other_node in self.nodes :
                if other_node != node :
                    dist [other_node] = 100000000

            for i in range (len(self.nodes) - 1):
                for edge in self.edges :
                    src, dest, cost = edge
                    if dist[src] + cost < dist[dest]:
                        dist[dest] = dist[src] + cost
                        if src == node :
                            next_hop[dest] = dest
                        elif src in next_hop :
                            next_hop[dest] = next_hop[src]
            self.print_routing_table(node, dist, next_hop
            print()
```

```
def print_routing_table(self, node, dist, next_ho
    print(f'Routing table for {node}:')
    print('Dest \t Cost \t Next Hop')
    for dest, cost in dist.items():
        print(f'{dest} \t {cost} \t
              {next_hop [dest]}')

def start(self):
    pass
```

PROGRAM 7B: DIJKSTRA'S ALGORITHM

CODE

```
import sys

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", dist[node])

    def minDistance(self, dist, sptSet):
        min = sys.maxSize
        for v in range(self.V):
            if dist[v] < min and sptSet[v] = Fals
                min = dist[v]
```

2

Scanned with CamScanner

```
            min_index = V
    return min_index

def dijkstra (self, src) :
    dist = [sys.maxsize] * 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)
```

```
if ((len(buffer.buffer) <= buffer.buffer_size)
    if j< len(data_to_send):
        buffer.buffer.append(data_to_send[j])
        j+ = 1
else
    if j < len(data_to_send):
        print("Data loss" + data_to_send[j]
        j+ = 1;
```

SPLASH

PROGRAM 8 : LEAKY BUCKET ALGORITHM

CODE

```
import os
clear = lambda : os.system('clear')

class Client :
    def _init_(self, rate=int, data=[]):
        self.rate = rate
        self.data = data

    def _str_(self):
        return str([str(self.rate),
                    str(self.data)])

class Buffer :

    def _init_(self, buffer_size=int, buffer=[
        self.buffer_size = buffer_size
        self.buffer = buffer

    def checkstate(self) :
        if len(self.buffer) == 0 :
            return True

    def _str_(self):
        return str([str(self.buffer_size),
                    str(self.buffer)]);

basestate = True
sec = 1
```

```
sentence = input("enter file name")
clientSocket.send(sentence.encode())
filecontents = clientSocket.recv(1024).decode
print("From Server:", filecontents)

clientSocket.close()
```