LAB ASSESSMENT – 3

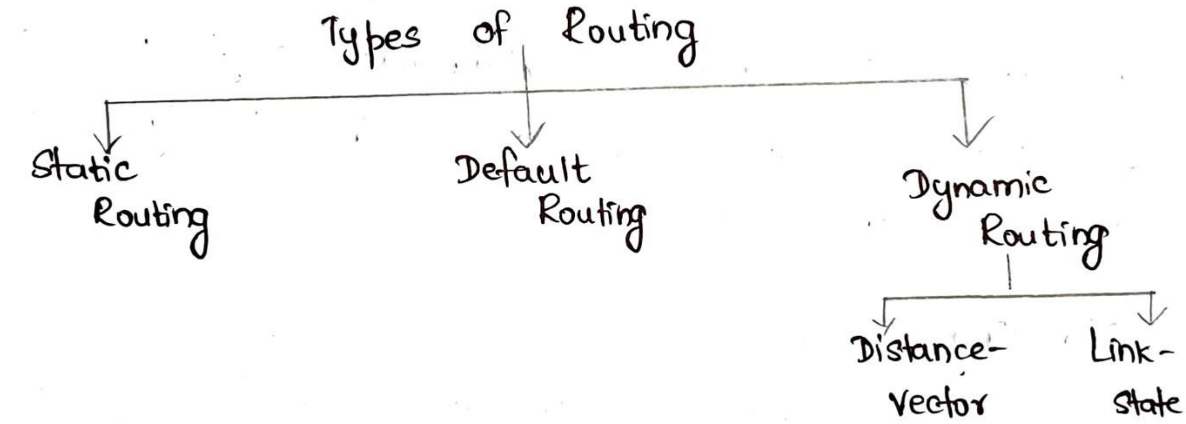
Link State Algorithm.

– Dijkstra Algorithm.

I. AIM:-

To implement the Dijkstra algorithm using C program for finding the shortest path between routers and to demonstrate the working of the algorithm.

II. Problem Analysis:-



LINK-STATE ROUTING:-

* Link state routing is the second family of routing protocols. While distance-vector routers use a distributed algorithm to compute their routing tables, link-state routing uses link-State routers to exchange messages that allow each router to learn the entire network topology.

* Based on this learned topology, each router is then able to compute its routing table by using a shortest path computation.

→ Link-State routing is a technique in which each router shares the knowledge of its neighbourhood with energy other router in the internetwork.

Three Keys to understand the Link-State Routing Algorithm:-

* **Knowledge about the neighbourhood:-**

Instead of sending its routing table, a router sends the information about its neighbourhood only. A router broadcast its identities and cost of the directly attached links to other routers.

* **Flooding:-**

Each router sends the information to every other router on the internet except its neighbours. This process is known as flooding. Every router that receives the packet sends the copies to all its neighbours. Finally each and every router receives a copy of the same information.

* **Information Sharing:-**

A router sends the information to every other router only when the change occurs in the information.

Link state Routing has two phases:-

Reliable Flooding:-

→ Initial State:- Each node knows the cost of its neighbours.

→ Final State :- Each node knows the entire graph.

Route Calculation:-

Each node uses Dijkstra's algorithm on the graph to calculate the optimal routes to all nodes.

Stepwise Explanation of Dijkstra's Algorithm:-

To find the shortest path, each node need to run the Dijkstra's algorithm. This algorithm uses the following steps:-

Step-1:-

The node is taken and chosen as a root node of the tree. This creates the tree with a single node and now set the total cost of each node to some value based on the information in Linkstate Database.

Step-2:-

Now the node selects one node among all nodes in the tree like structure, which is nearest to the root, and adds this to the tree. The shape of the tree gets changed.

Step-3:-

After this node is added to the tree, the cost of all the nodes not in the tree needs to be updated because the paths may have been changed.

Step-4:-

The node repeats Step 2 and Step 3 until all the nodes are added in the tree.

Algorithm:-

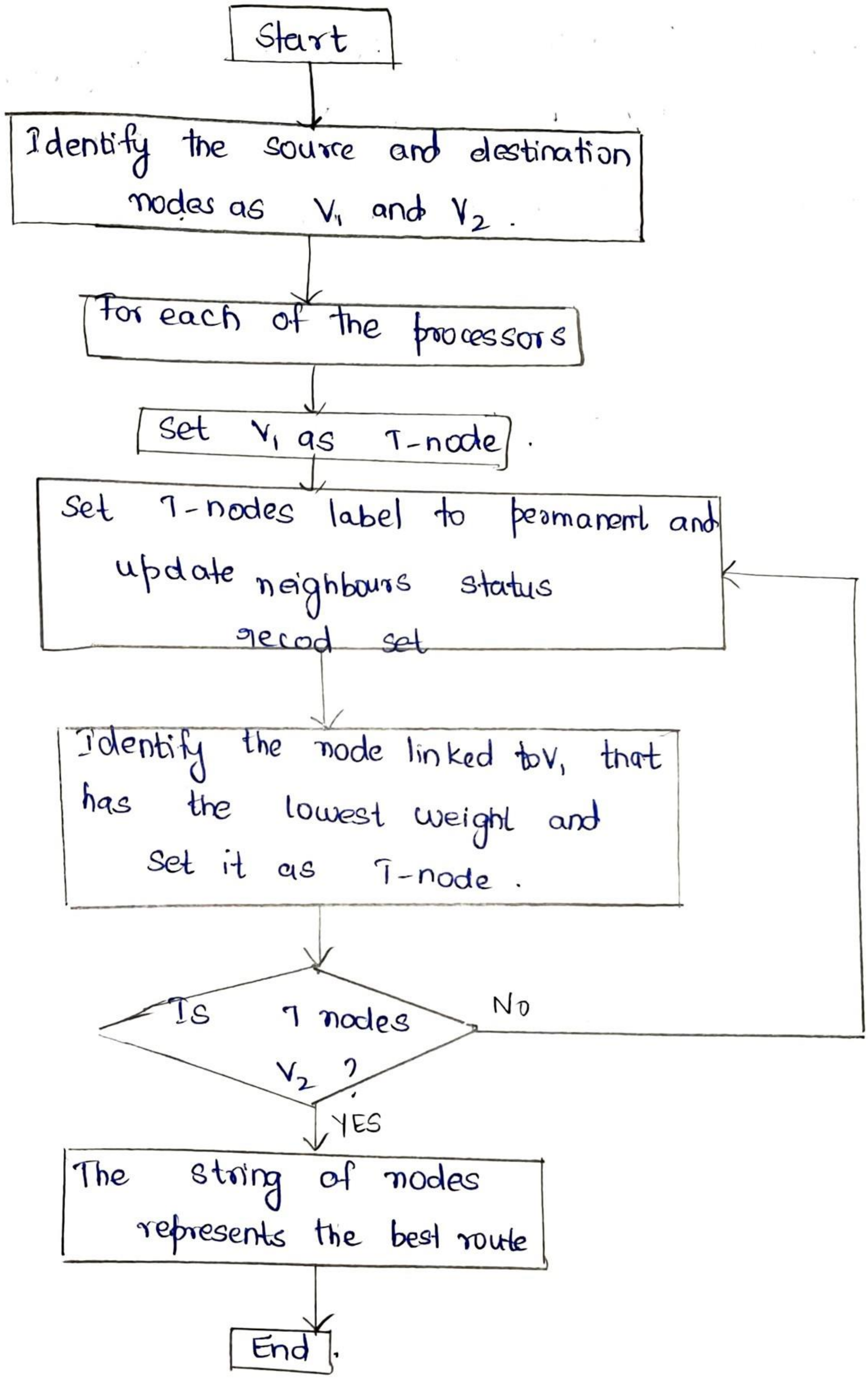
Initialization:-

N = {A} // A is a root node.

for all nodes v
if v adjacent to A
then D(v) = c(A, v)
else D(v) = infinity
loop
find w not in N such that D(w) is a minimum
Add N to N
update D(v) for all v adjacent to W and not in N:
D(v) = min (D(v), D(W) + c(W, v))
until all nodes in N.

* c(i, j) : link cost from node i to node j.
If i and j are not directly linked then c(i, j) = ∞

III FLOW CHART



Dijisktra's Algorithm

```
maxint = 100
class Graph():

    def __init__(self, vertices):
        self.V = vertices   ## initializing the vertex
        self.graph = [      ## initializing the graph
                [0 for column in range(vertices)]
                for row in range(vertices)]

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

    def minimum_distance(self, dist, sptSet):
        min = maxint
        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 = [maxint] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for i in range(self.V):
            u = self.minimum_distance(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[
                        dist[v] = dist[u] + self.graph[u][v]
        self.printSolution(dist)

if __name__ == "__main__":
    n = int(input("Enter the vertices : "))
    initial_vertex = 0
    g = Graph(n)        ## initializing an empty graph
    print("\nEnter the adjacency matrix")
    adjacency_matrix = []
    temp = []


    for j in range(n):
        temp = list(map(int,input().split()))
        adjacency_matrix.append(temp)

    g.graph = adjacency_matrix
    g.dijkstra(initial_vertex)
```

Output

```
Enter the vertices : 5

Enter the adjacency matrix
0 5 2 3 0
5 0 4 0 3
2 4 0 0 4
3 0 0 0 0
0 3 4 0 0
Vertex  Distance from Source
0    0
1    5
2    2
3    3
4    6

***Repl Closed***
```