CSI 2007 – DATA COMMUNICATION AND NETWORKS

CODE ➔ [LAB DA 4.ipynb - Colaboratory (google.com)]

MAY 13, 2021

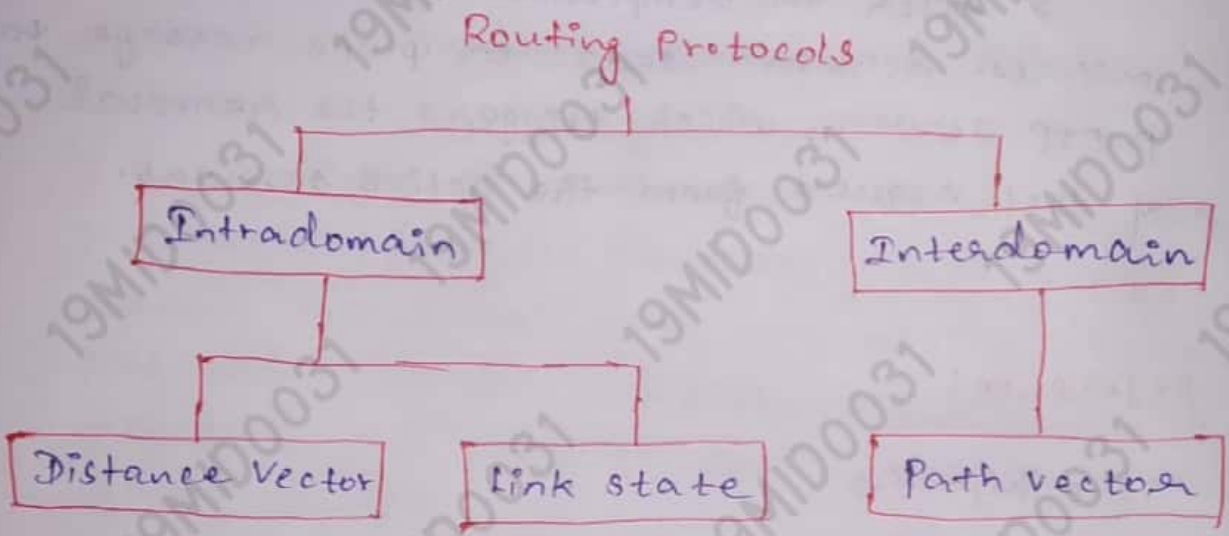
S.THARUN
19MID0031

LINK STATE ROUTING - DIJIKSTRA'S ALGORITHM

Aim :

To understand and implement the Link state routing protocol using Dijikstra's Algorithm and analyze them.

PROBLEM ANALYSIS



* In Link state Routing, each node in the network knows entire network topology such as:
  - → the list of nodes and links
  - → how they are connected
  - → cost
  - → links up/down

* Each node uses Dijikstra's algorithm to build a shortest path tree.

Properties:

* OSPF, ISIS
* Routers communicate with all other routers exchanging link-state information to build a topology of entire network
* Link state ⇒ interface connections
  (or)
  links to other routers/network
* Best for
  - large, hierarchical networks
  - advanced administrator knowledge
  - convergence time is crucial.
* Routers have a complete view of the network, knowledge of the entire topology
* Send triggered partial updates

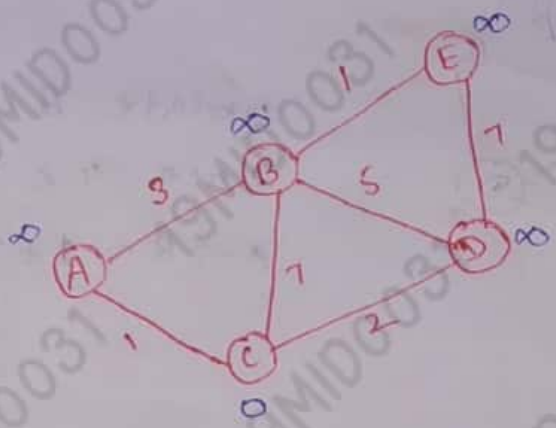
DIJKSTRA'S ALGORITHM

* Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in graph, which may represent, for example road networks, routing networks etc.
* It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

LINK STATE ROUTING ALGORITHM - DIJKSTRA

* ensures every router has complete information about all other routers in the network and the traffic states of network

* Dijkstra alg → shortest path b/w source to every other node.

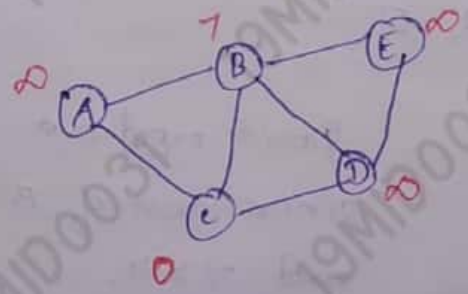


C → source node

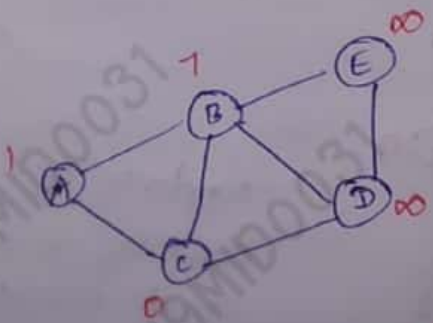
Distance from C to C → 0

all other distance unknown → ∞

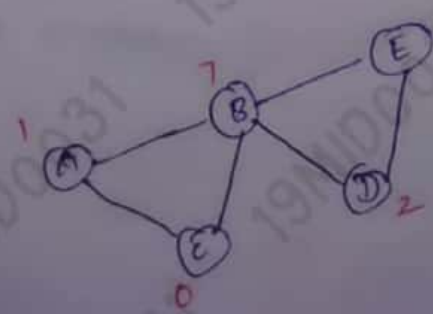
* Neighbours of C → A, B, D



update B ∵ 7 < ∞

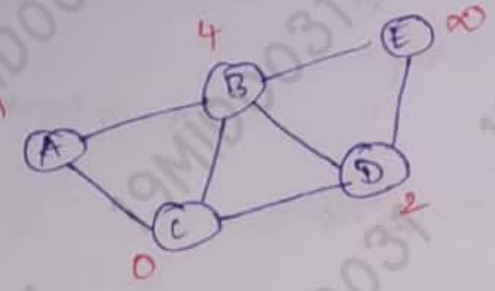


update A ∵ 1 < ∞



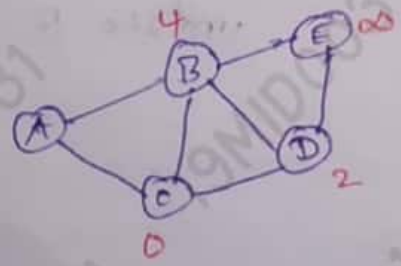
update C ∵ 2 < ∞

* Now c is visited and all others are unvisited
Let's move to shortest distance value, i.e) A
Now A is current node.
Neighbours of A → C, B
But c is already visited
Let's go to B

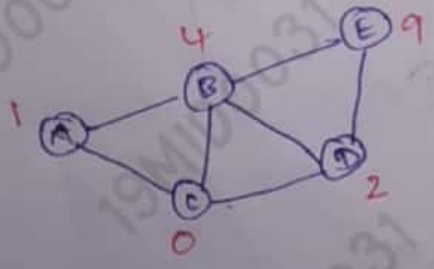


update B
since 4 < 7
A → B = 1 + 3 = 4

* visited = {c, A}
unvisited = {B, D, E}
small distance among unvisited ⇒ D
So Now D is current node.
Neighbours of D → B, E



Don't update B.
since cost of B = 2+5 = 7
and 7 > 4.



update E
cost of E = 2 + 7 = 9
and 9 < ∞
So update E.

visited = {C, A, D}

unvisited = {B, E}

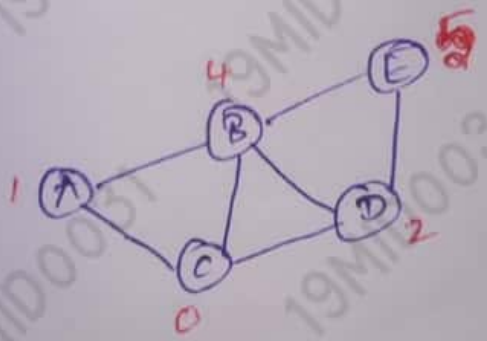
shortest among unvisited ⇒ B

So Now current node is B

Neighbours of B ⇒ {A, C, D, E}

But {A, C, D} are visited

So Let's check the unvisited node E



cost of B = 4 + 1 = 5

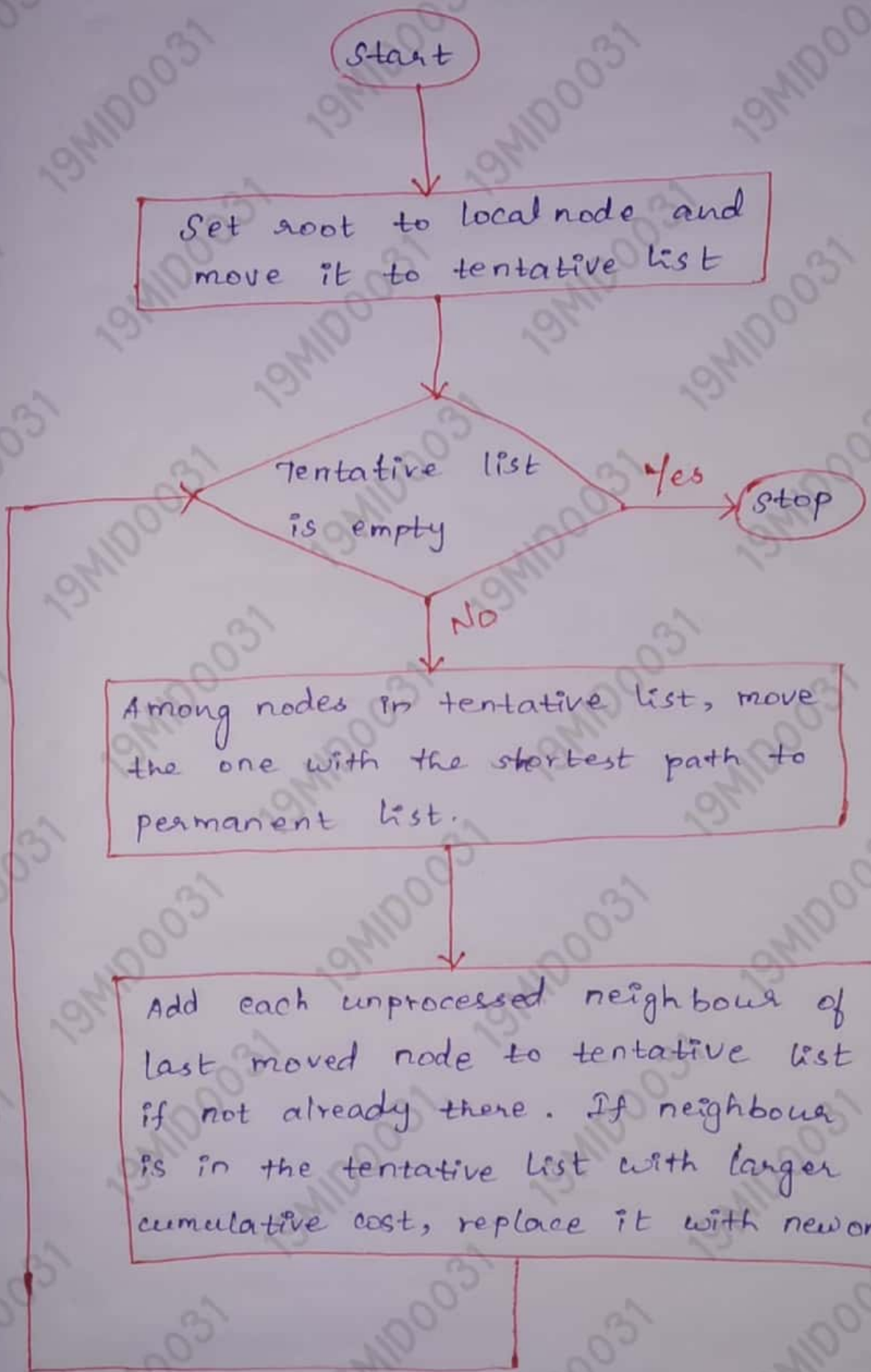
and 5 < 9.

So update E.

Pseudocode :

```
function Dijkstra (Graph, source):
    for each vertex v in Graph:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist [source] ← 0
    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q
        for each neighbour v of u:
            alt ← dist[u] + length [u, v]
            if alt < dist [v]:
                dist[v] ← alt
                prev[v] ← u
    return dist[], prev[]
```

Flowchart - Link State Routing :



CODE:

FINDING SHORTEST PATH USING DIJKSTRA'S ALGORITHM & FORMING ROUTING TABLE

```
from pandas import DataFrame
```

```
graph = {'A':{'C':1,'B':3},
         'B':{'A':3,'C':7,'D':5,'E':1},
         'C':{'A':1,'B':7,'D':2},
         'D':{'B':5,'C':2,'E':7},
         'E':{'B':1,'D':7}
        }

nodes = list(graph.keys())
RoutingTable = []

for p,q in enumerate(nodes):

    visited = []
    unvisited = nodes.copy()
    next_node = len(nodes)*['']

    inf = float('inf')
    shortest_distance = len(nodes)*[inf]

    root_node = q
    current_node = q
    shortest_distance[ord(current_node)-65] = 0
```

```
    while True:
        for i in list(graph[current_node].keys()):
            if i not in visited:
                if shortest_distance[ord(current_node)-65]+graph[current_node][i] < shortest_distance[ord(i)-65]:
                    shortest_distance[ord(i)-65] = shortest_distance[ord(current_node)-65]+graph[current_node][i]
                    if current_node != root_node:
                        next_node[ord(i)-65] = current_node

        visited.append(current_node)
        unvisited.remove(current_node)

        if len(unvisited) == 0:
            break

        unvstd_aasci = [ord(x) for x in unvisited]
        min_value = min([shortest_distance[j-65] for j in unvstd_aasci])
        min_index = [j for j,x in enumerate(shortest_distance) if x == min_value]
        for j in min_index:
            if chr(65+j) in unvisited:
                current_node = chr(65+j)
                break

    for i in range(len(next_node)):
        if next_node[i] == root_node:
            next_node[i] = ''
```

```
    RoutingTable.append(DataFrame({'To':list(graph.keys()),'Cost':shortest_distance,'Next':next_node}))

    print(f"\n\nRouting table for {root_node}")
    display(RoutingTable[-1])

print(f"\n\nAvailable nodes ==> {nodes}")
```

TO FIND OPTIMAL PATH FROM ROUTING TABLE

```
start = input("\nEnter the start node : ")
while start not in nodes:
    print("Invalid Node....Try again")
    start = input("Enter the start node : ")


dest  = input("\nEnter the Destination node : ")
while dest not in nodes:
    print("Invalid Node....Try again")
    dest = input("Enter the Destination node : ")


index = ord(start)-65
df = RoutingTable[index]
path = []
path.append(start)
temp = dest
while df[df['To'] == temp]['Next'].values[0] != '':
    path.insert(1,df[df['To'] == temp]['Next'].values[0])
    temp = df[df['To'] == temp]['Next'].values[0]

print("\nOptimal path : ",end ="")
for i in range(len(path)):
    print(path[i],end = " ==> ")
print(dest)
```

CLICK ON THE LINK IN FIRST PAGE TO EXECUTE THE CODE

OUTPUT:

```
Routing table for A
```

	To	Cost	Next
0	A	0	
1	B	3	
2	C	1	
3	D	3	C
4	E	4	B

```
Routing table for B
```

	To	Cost	Next
0	A	3	
1	B	0	
2	C	4	A
3	D	5	
4	E	1	

Routing table for C

	To	Cost	Next
0	A	1	
1	B	4	A
2	C	0	
3	D	2	
4	E	5	B

Routing table for D

	To	Cost	Next
0	A	3	C
1	B	5	
2	C	2	
3	D	0	
4	E	6	B

```
Routing table for E
```

	To	Cost	Next
0	A	4	B
1	B	1	
2	C	5	A
3	D	6	B
4	E	0	

```
Available nodes ==> ['A', 'B', 'C', 'D', 'E']

Enter the start node : f
Invalid Node....Try again
Enter the start node : C

Enter the Destination node : e
Invalid Node....Try again
Enter the Destination node : E

Optimal path : C ==> A ==> B ==> E
```

ALL ROUTING TABLE HAS BEEN DISPLAYED ON ACCOUNT OF ALL TEST CASES.

IF INVALID NODES ARE GIVEN AS INPUT, THE RESPECTIVE OUTPUT IS SHOWN

Result and Analysis:

Link state Routing protocol has been implemented in python using Dijkstra's Algorithm. The pros and cons of the protocols have been clearly discussed in this Assignment. This protocol is prefered and advantageous if the network is large sized.