Unit 3 - Routing Techniques

In network layer we deals with packets, data link had frames . ...

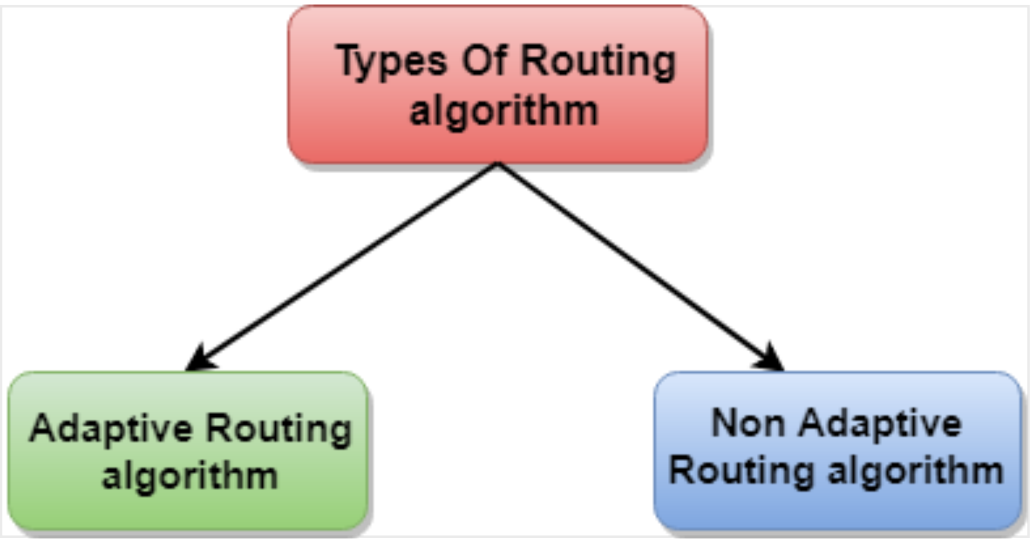
Routing algorithm

- In order to transfer the packets from source to the destination, the network layer must determine the best route through which packets can be transmitted.
- Whether the network layer provides datagram service or virtual circuit service, the main job of the network layer is to provide the best route. The routing protocol provides this job.
- The routing protocol is a routing algorithm that provides the best path from the source to the destination. The best path is the path that has the "least-cost path" from source to the destination.
- Routing is the process of forwarding the packets from source to the destination but the best route to send the packets is determined by the routing algorithm.

Classification of a Routing algorithm

The Routing algorithm is divided into two categories:

- Adaptive Routing algorithm
- Non-adaptive Routing algorithm


Types Of Routing algorithm
Adaptive Routing algorithm
Non Adaptive Routing algorithm

Adaptive Routing algorithm

- An adaptive routing algorithm is also known as dynamic routing algorithm.
- This algorithm makes the routing decisions based on the topology and network traffic.

- The main parameters related to this algorithm are hop count, distance and estimated transit time.

An adaptive routing algorithm can be classified into three parts:

- **Centralized algorithm:** It is also known as global routing algorithm as it computes the least-cost path between source and destination by using complete and global knowledge about the network. This algorithm takes the connectivity between the nodes and link cost as input, and this information is obtained before actually performing any calculation. **Link state algorithm** is referred to as a centralized algorithm since it is aware of the cost of each link in the network.
- **Isolation algorithm:** It is an algorithm that obtains the routing information by using local information rather than gathering information from other nodes.
- **Distributed algorithm:** It is also known as decentralized algorithm as it computes the least-cost path between source and destination in an iterative and distributed manner. In the decentralized algorithm, no node has the knowledge about the cost of all the network links. In the beginning, a node contains the information only about its own directly attached links and through an iterative process of calculation computes the least-cost path to the destination. A Distance vector algorithm is a decentralized algorithm as it never knows the complete path from source to the destination, instead it knows the direction through which the packet is to be forwarded along with the least cost path.

Non-Adaptive Routing algorithm

- Non Adaptive routing algorithm is also known as a static routing algorithm.
- When booting up the network, the routing information stores to the routers.
- Non Adaptive routing algorithms do not take the routing decision based on the network topology or network traffic.

The Non-Adaptive Routing algorithm is of two

types:

1) . Flooding –

- Requires no network information like topology, load condition, cost of diff. paths
- Every incoming packet to a node is sent out on every outgoing like except the one it arrived on.
- For Example in the above figure
  - An incoming packet to (1) is sent out to (2),(3)
  - from (2) is sent to (6),(4), and from (3) it is sent to (4),(5)
  - from (4) it is sent to (6),(5),(3), from (6) it is sent to (2),(4),(5), from (5) it is sent to (4),(3)

Characteristics –

- All possible routes between Source and Destination are tried. A packet will always get through if the path exists
- As all routes are tried, there will be at least one route which is the shortest
- All nodes directly or indirectly connected are visited

Limitations –

- Flooding generates a vast number of duplicate packets
- Suitable damping mechanism must be used

Hop-Count –

- A hop counter may be contained in the packet header which is decremented at each hop.
  with the packet being discarded when the counter becomes zero
- The sender initializes the hop counter. If no estimate is known, it is set to the full diameter of the subnet.
- Keep track of the packets which are responsible for flooding using a sequence number. Avoid sending them out a second time.

Selective Flooding: Routers do not send every incoming packet out on every line, only on those lines that go in approximately in the direction of the destination.

Advantages of Flooding :

- Highly Robust, emergency or immediate messages can be sent (eg military applications)
- Set up the route in virtual circuit
- Flooding always chooses the shortest path

- Broadcast messages to all the nodes

Disadvantages of Flooding :

- Network congestion: Flooding can cause a significant amount of traffic in the network, leading to congestion. This can result in slower network speeds and delays in delivering data packets.
- Wastage of network resources: Flooding uses a lot of network resources, including bandwidth and processing power, to deliver packets. This can result in the wastage of valuable network resources and reduce the overall efficiency of the network.
- Security risks: Flooding can be used as a tool for launching various types of attacks, including denial of service (DoS) attacks. Attackers can flood the network with data packets, which can overload the network and cause it to crash.
- Inefficient use of energy: Flooding can result in an inefficient use of energy in wireless networks. Since all nodes receive every packet, even if they are not the intended recipient, they will still need to process it, which can waste energy and reduce the overall battery life of mobile devices.
- Difficulty in network troubleshooting: Flooding can make it difficult to troubleshoot network issues. Since packets are sent to all nodes, it can be challenging to isolate the cause of a problem when it arises.

2). Dijkstra's Algorithm

The following tutorial will teach us about Dijkstra's Shortest Path Algorithm. We will understand the working of Dijkstra's Algorithm with a stepwise graphical explanation.

We will cover the following:

- A Brief Overview of the Fundamental Concepts of Graph
- Understand the Use of Dijkstra's Algorithm
- Understand the Working of the Algorithm with a Step-by-Step Example

So, let's get started.

A Brief Introduction to Graphs

Graphs are non-linear data structures representing the "connections" between the elements. These elements are known as the **Vertices**, and the lines or arcs that connect any two vertices in the graph are known as the **Edges**. More formally, a Graph comprises **a set of Vertices (V)** and **a set of Edges (E)**. The Graph is denoted by **G(V, E)**.

Components of a Graph

1. **Vertices:**Vertices are the basic units of the graph used to represent real-life objects, persons, or entities. Sometimes, vertices are also known as Nodes.
2. **Edges:**Edges are drawn or used to connect two vertices of the graph. Sometimes, edges are also known as Arcs.

The following figure shows a graphical representation of a Graph:

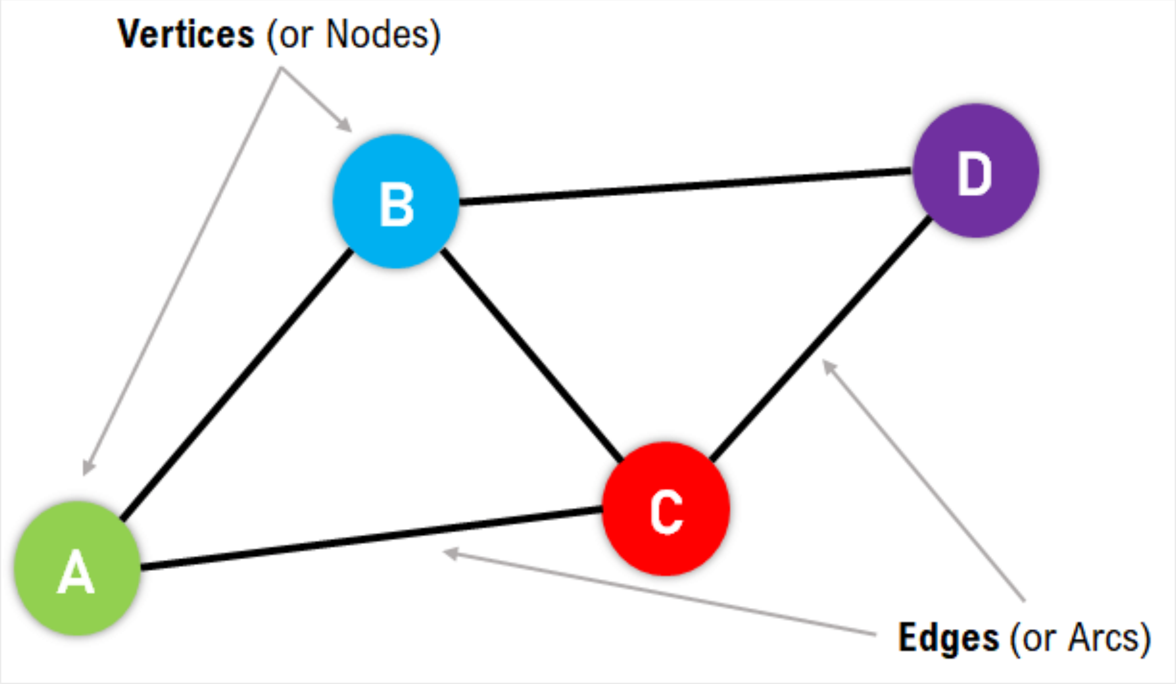


Figure 1: Graphical Representation of a Graph

In the above figure, the Vertices/Nodes are denoted with Colored Circles, and the Edges are denoted with the lines connecting the nodes.

Types of Graphs

The Graphs can be categorized into two types:

1. Undirected Graph
2. Directed Graph

Undirected Graph: A Graph with edges that do not have a direction is termed an Undirected Graph. The edges of this graph imply a two-way relationship in which each edge can be traversed in both directions. The following figure displays a simple undirected graph with four nodes and five edges.

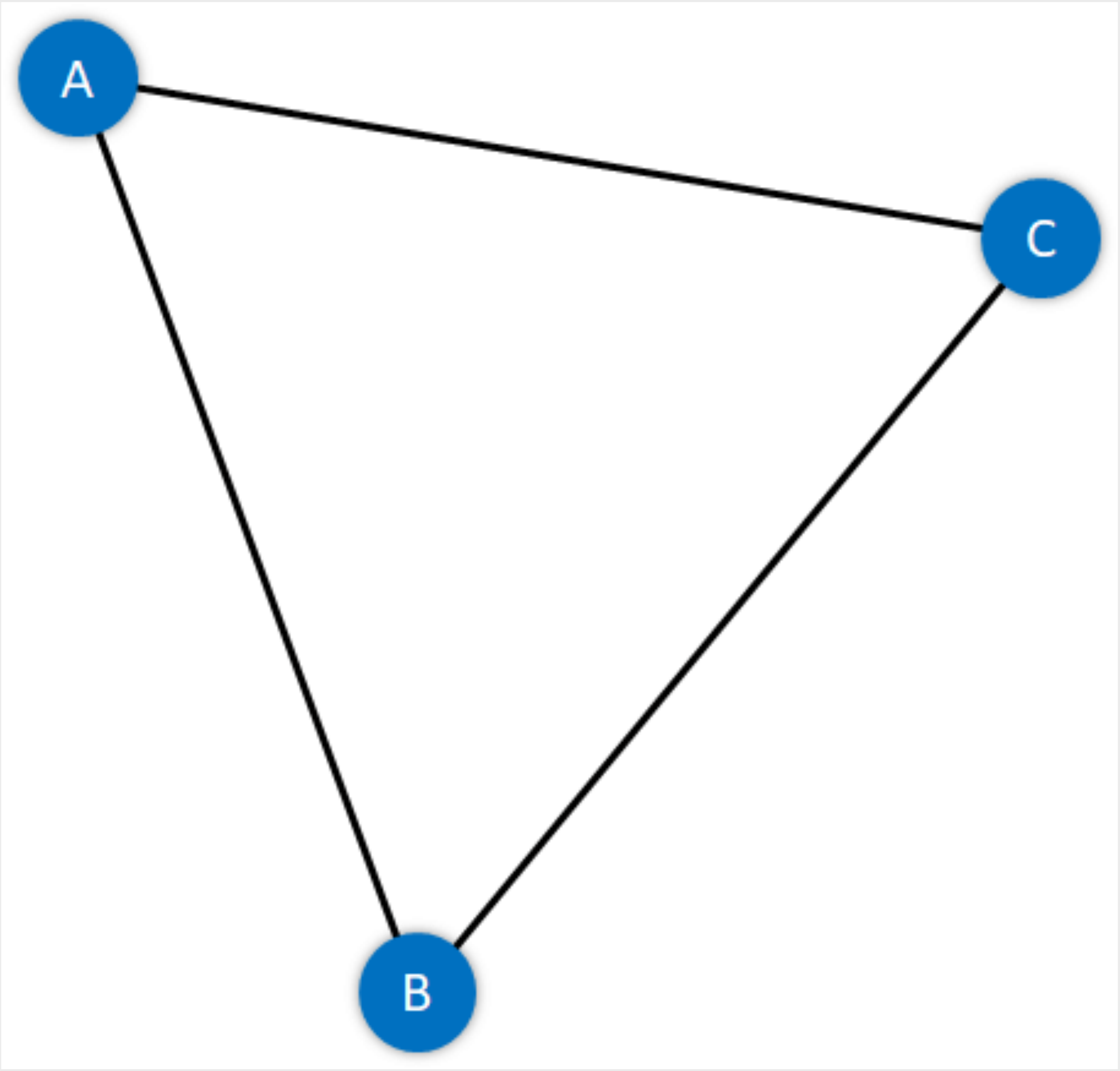


Figure 3: A Simple Undirected Graph

Directed Graph: A Graph with edges with direction is termed a Directed Graph. The edges of this graph imply a one-way relationship in which each edge can only be traversed in a single direction. The following figure displays a simple directed graph with four nodes and five edges.

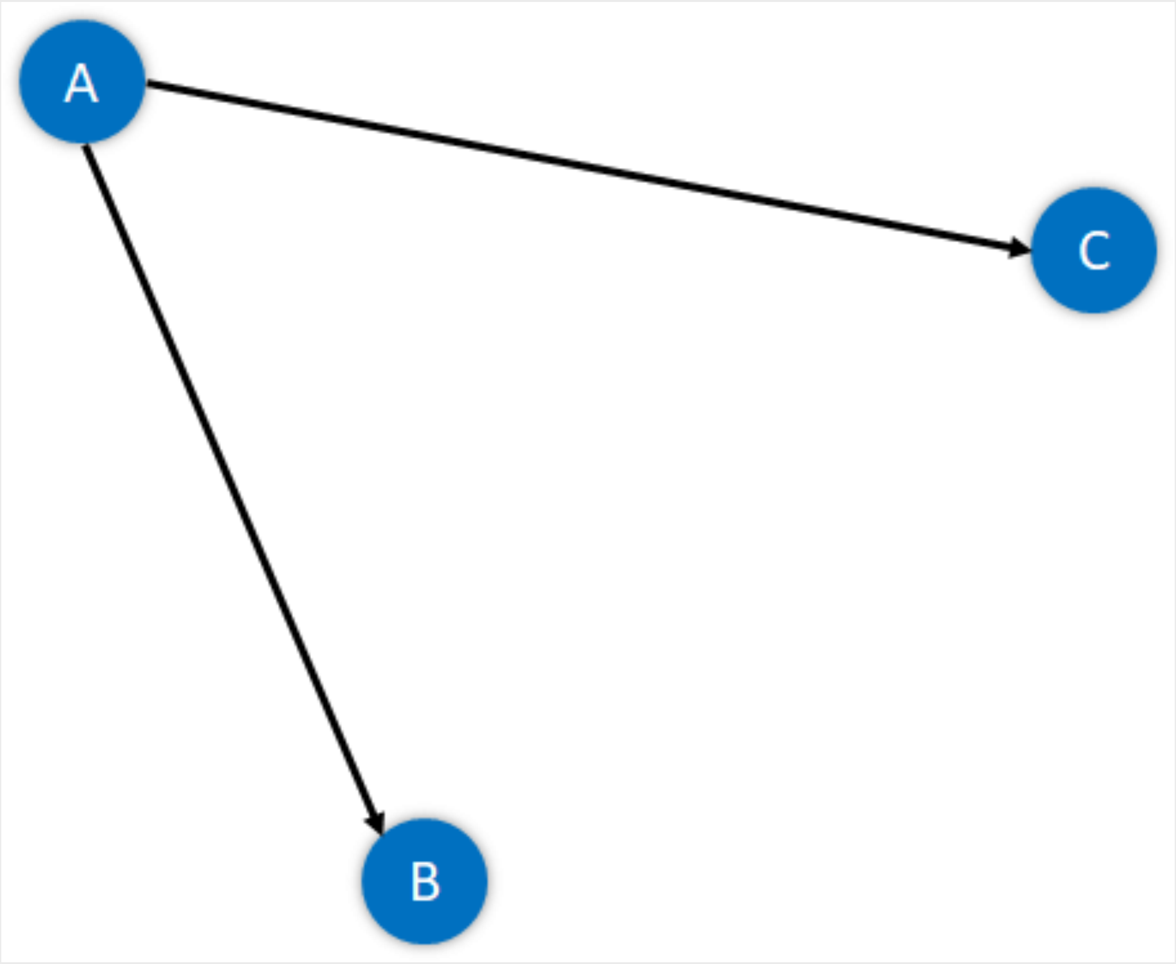


Figure 4: A Simple Directed Graph

The absolute length, position, or orientation of the edges in a graph illustration characteristically does not have meaning. In other words, we can visualize the same graph in different ways by rearranging the vertices or distorting the edges if the underlying structure of the graph does not alter.

What are Weighted Graphs?

A Graph is said to be Weighted if each edge is assigned a 'weight'. The weight of an edge can denote distance, time, or anything that models the 'connection' between the pair of vertices it connects. For instance, we can observe a blue number next to each edge in the following figure of the Weighted Graph. This number is utilized to signify the weight of the corresponding edge.

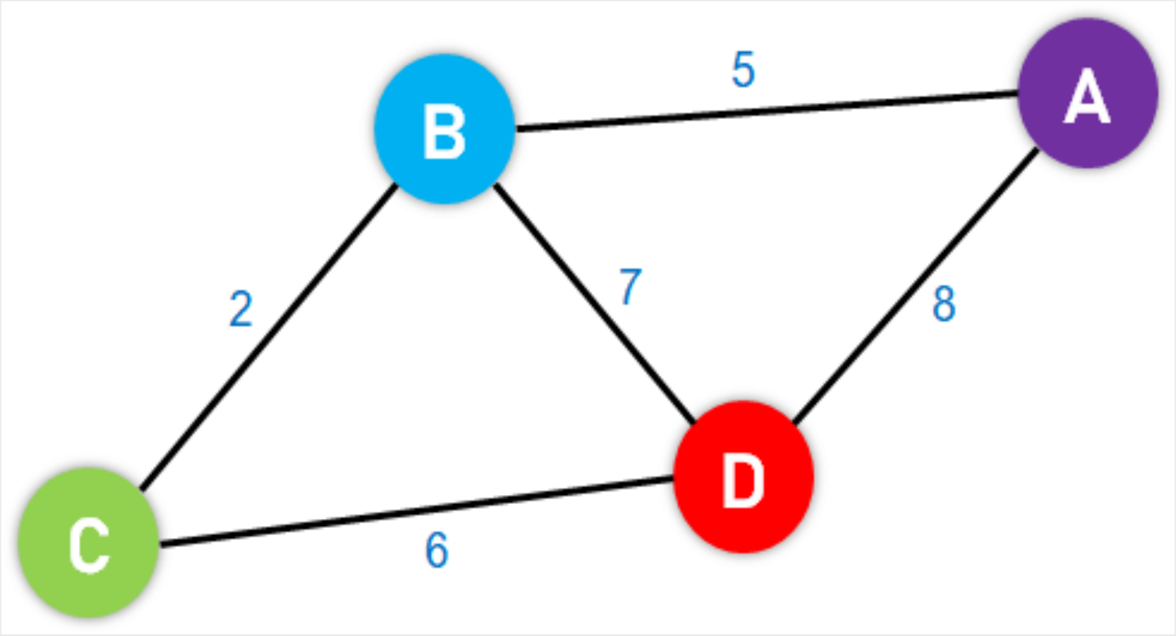


Figure 5: An Example of a Weighted Graph

An Introduction to Dijkstra's Algorithm

Now that we know some basic Graphs concepts let's dive into understanding the concept of Dijkstra's Algorithm.

Ever wondered how does Google Maps finds the shortest and fastest route between two places?

Well, the answer is **Dijkstra's Algorithm**. **Dijkstra's Algorithm** is a Graph algorithm **that finds the shortest path** from a source vertex to all other vertices in the Graph (single source shortest path). It is a type of Greedy Algorithm that only works on Weighted Graphs having positive weights. The time complexity of Dijkstra's Algorithm is $\mathbf{O(V^2)}$ with the help of the adjacency matrix representation of the graph. This time complexity can be reduced to **O((V + E) log V)** with the help of an adjacency list representation of the graph, where **V** is the number of vertices and **E** is the number of edges in the graph.

History of Dijkstra's Algorithm

Dijkstra's Algorithm was designed and published by **Dr. Edsger W. Dijkstra**, a Dutch Computer Scientist, Software Engineer, Programmer, Science Essayist, and Systems Scientist.

During an Interview with Philip L. Frana for the Communications of the ACM journal in the year 2001, Dr. Edsger W. Dijkstra revealed:

"What is the shortest way to travel from Rotterdam to Groningen, in general: from given city to given city? It is the algorithm for the shortest path, which I designed in about twenty minutes. One morning I was shopping in Amsterdam with my young fiancée, and

tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in '59, three years later. The publication is still readable, it is, in fact, quite nice. One of the reasons that it is so nice was that I designed it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities. Eventually, that algorithm became to my great amazement, one of the cornerstones of my fame."
Dijkstra thought about the shortest path problem while working as a programmer at the Mathematical Centre in Amsterdam in 1956 to illustrate the capabilities of a new computer known as ARMAC. His goal was to select both a problem and a solution (produced by the computer) that people with no computer background could comprehend. He developed the shortest path algorithm and later executed it for ARMAC for a vaguely shortened transportation map of 64 cities in the Netherlands (64 cities, so 6 bits would be sufficient to encode the city number). A year later, he came across another issue from hardware engineers operating the next computer of the institute: Minimize the amount of wire required to connect the pins on the machine's back panel. As a solution, he re-discovered the algorithm called Prim's minimal spanning tree algorithm and published it in the year 1959.

Fundamentals of Dijkstra's Algorithm

The following are the basic concepts of Dijkstra's Algorithm:

1. Dijkstra's Algorithm begins at the node we select (the source node), and it examines the graph to find the shortest path between that node and all the other nodes in the graph.
2. The Algorithm keeps records of the presently acknowledged shortest distance from each node to the source node, and it updates these values if it finds any shorter path.
3. Once the Algorithm has retrieved the shortest path between the source and another node, that node is marked as 'visited' and included in the path.
4. The procedure continues until all the nodes in the graph have been included in the path. In this manner, we have a path connecting the source node to all other nodes, following the shortest possible path to reach each node.

Understanding the Working of Dijkstra's Algorithm

A **graph** and **source vertex** are requirements for Dijkstra's Algorithm. This Algorithm is established on Greedy Approach and thus finds the locally optimal choice (local minima in this case) at each step of the Algorithm.

Each Vertex in this Algorithm will have two properties defined for it:

1. Visited Property
2. Path Property

Let us understand these properties in brief.

Visited Property:

1. The 'visited' property signifies whether or not the node has been visited.
2. We are using this property so that we do not revisit any node.
3. A node is marked visited only when the shortest path has been found.

Path Property:

1. The 'path' property stores the value of the current minimum path to the node.
2. The current minimum path implies the shortest way we have reached this node till now.
3. This property is revised when any neighbor of the node is visited.
4. This property is significant because it will store the final answer for each node.

Initially, we mark all the vertices, or nodes, unvisited as they have yet to be visited. The path to all the nodes is also set to infinity apart from the source node. Moreover, the path to the source node is set to zero (0).

We then select the source node and mark it as visited. After that, we access all the neighboring nodes of the source node and perform relaxation on every node. Relaxation is the process of lowering the cost of reaching a node with the help of another node.

In the process of relaxation, the path of each node is revised to the minimum value amongst the node's current path, the sum of the path to the previous node, and the path from the previous node to the current node.

Let us suppose that p[n] is the value of the current path for node n, p[m] is the value of the path up to the previously visited node m, and w is the weight of the edge between the current node and previously visited one (edge weight between n and m).

In the mathematical sense, relaxation can be exemplified as:

p[n] = minimum(p[n], p[m] + w)

We then mark an unvisited node with the least path as visited in every subsequent step and update its neighbor's paths.

We repeat this procedure until all the nodes in the graph are marked visited.

Whenever we add a node to the visited set, the path to all its neighboring nodes also changes accordingly.

If any node is left unreachable (disconnected component), its path remains 'infinity'. In case the source itself is a separate component, then the path to all other nodes remains 'infinity'.

Understanding Dijkstra's Algorithm with an Example

The following is the step that we will follow to implement Dijkstra's Algorithm:

Step 1: First, we will mark the source node with a current distance of 0 and set the rest of the nodes to INFINITY.

Step 2: We will then set the unvisited node with the smallest current distance as the current node, suppose X.

Step 3: For each neighbor N of the current node X: We will then add the current distance of X with the weight of the edge joining X-N. If it is smaller than the current distance of N, set it as the new current distance of N.

Step 4: We will then mark the current node X as visited.

Step 5: We will repeat the process from **'Step 2'** if there is any node unvisited left in the graph.

Let us now understand the implementation of the algorithm with the help of an example:

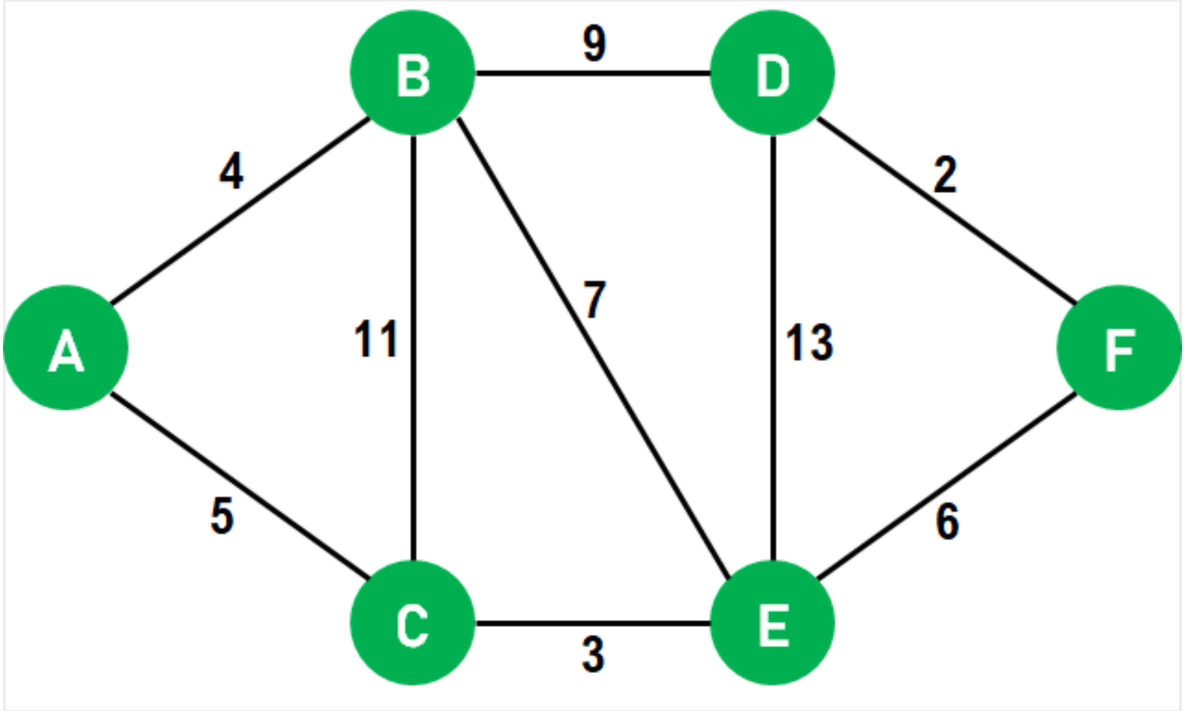


Figure 6: The Given Graph

1. We will use the above graph as the input, with node **A** as the source.
2. First, we will mark all the nodes as unvisited.
3. We will set the path to **0** at node **A** and **INFINITY** for all the other nodes.
4. We will now mark source node **A** as visited and access its neighboring nodes.
   Note: We have only accessed the neighboring nodes, not visited them.
5. We will now update the path to node **B** by **4** with the help of relaxation because the path to node **A** is **0** and the path from node **A** to **B** is **4**, and the **minimum((0 + 4), INFINITY)** is **4**.
6. We will also update the path to node **C** by **5** with the help of relaxation because the path to node **A** is **0** and the path from node **A** to **C** is **5**, and the **minimum((0 + 5), INFINITY)** is **5**. Both the neighbors of node **A** are now relaxed; therefore, we can move ahead.
7. We will now select the next unvisited node with the least path and visit it. Hence, we will visit node **B** and perform relaxation on its unvisited neighbors. After performing relaxation, the path to node **C** will remain **5**, whereas the path to node **E** will become **11**, and the path to node **D** will become **13**.
8. We will now visit node **E** and perform relaxation on its neighboring nodes **B, D**, and **F**. Since only node **F** is unvisited,

it will be relaxed. Thus, the path to node **B** will remain as it is, i.e., **4**, the path to node **D** will also remain **13**, and the path to node **F** will become **14 (8 + 6)**.

9. Now we will visit node **D**, and only node **F** will be relaxed. However, the path to node **F** will remain unchanged, i.e., **14**.
10. Since only node **F** is remaining, we will visit it but not perform any relaxation as all its neighboring nodes are already visited.
11. Once all the nodes of the graphs are visited, the program will end.

Hence, the final paths we concluded are:

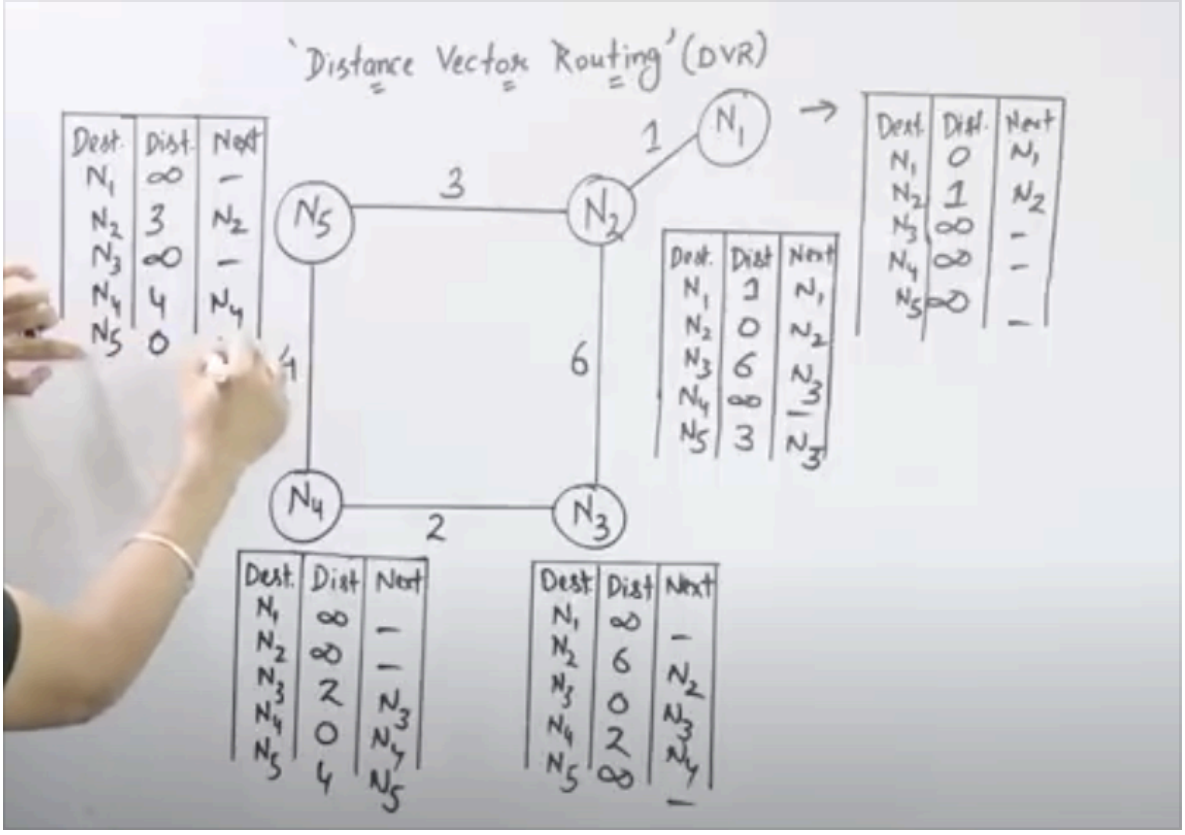
1. A = 0
2. B = 4 (A -> B)
3. C = 5 (A -> C)
4. D = 4 + 9 = 13 (A -> B -> D)
5. E = 5 + 3 = 8 (A -> C -> E)
6. F = 5 + 3 + 6 = 14 (A -> C -> E -> F)

Adaptive Routing Algorithms are of 3 types

-> **Distance Vector Routing**
-> Link Vector Routing
-> **Hierarchical Routing**

Distance Vector Routing

-> Here we have 5 routers connected to each other with some weight like graphs
-> will maintain routing table for each of them
-> first they send hello msg to their neighbor and calculate the initial distance vector table with neighbors and infinity for other then neighbors



'Distance Vector Routing' (DVR)

Dest.	Dist.	Next
N_1	∞	–
N_2	3	N_2
N_3	∞	–
N_4	4	N_4
N_5	0	

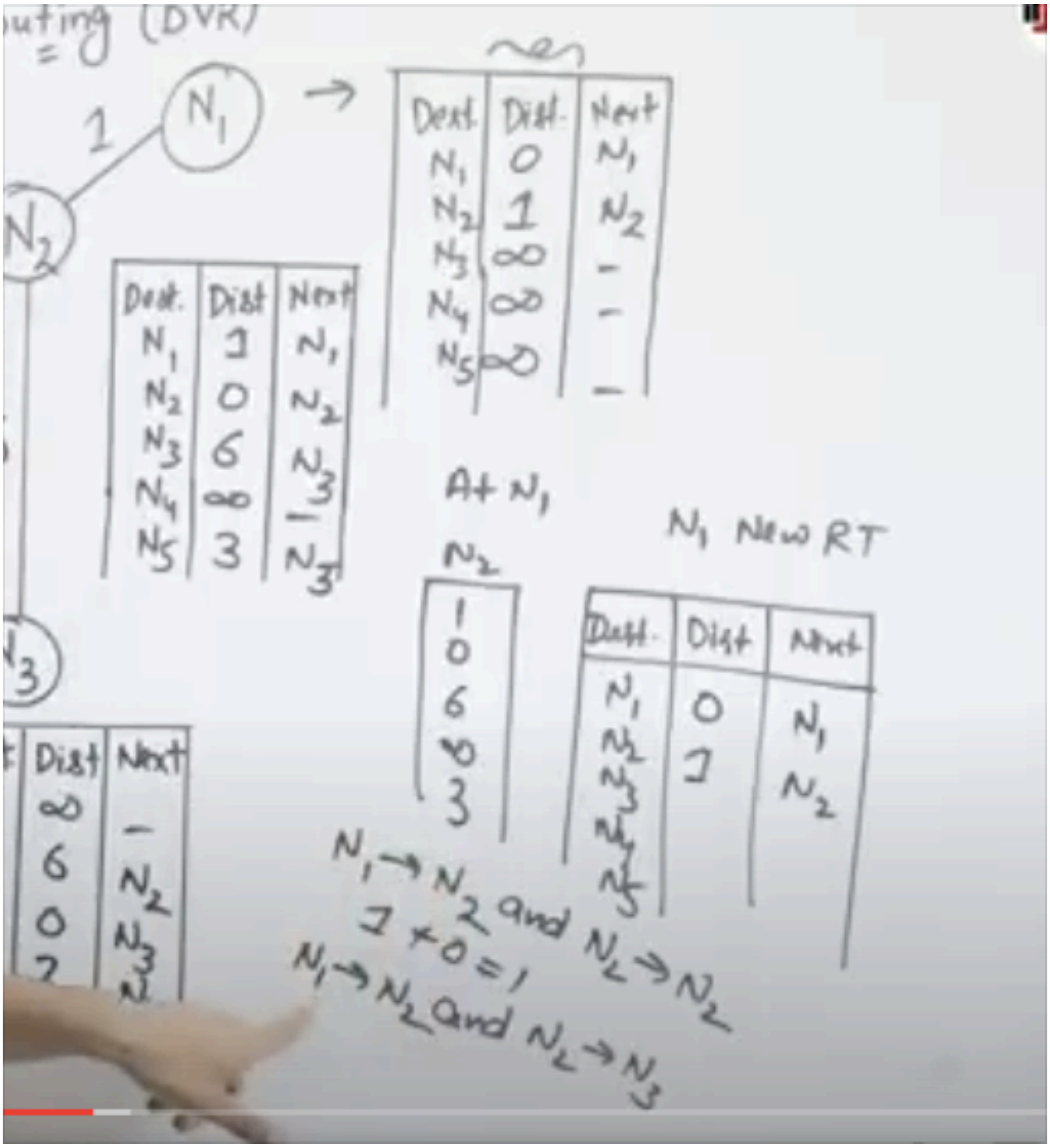
Dest.	Dist.	Next
N_1	0	N_1
N_2	1	N_2
N_3	∞	–
N_4	∞	–
N_5	∞	–

Dest.	Dist	Next
N_1	1	N_1
N_2	0	N_2
N_3	6	N_3
N_4	∞	–
N_5	3	N_3

Dest.	Dist	Next
N_1	∞	–
N_2	∞	–
N_3	2	N_3
N_4	0	N_4
N_5	4	N_5

Dest.	Dist	Next
N_1	∞	–
N_2	6	N_2
N_3	0	N_3
N_4	2	N_4
N_5	∞	–

-> then we will share the only distance vector column with neighbors
-> now in next step each of router will again update their routing tables according to their neighbors



Dest.	Dist.	Next
N1	0	N1
N2	1	N2
N3	∞	-
N4	∞	-
N5	∞	-

Dest.	Dist	Next
N1	1	N1
N2	0	N2
N3	6	N3
N4	∞	-
N5	3	N3

Dist	Next
∞	-
6	N2
0	N3
7	N

At N1

N2
1
0
6
∞
3

N1 New RT

Dest.	Dist	Next
N1	0	N1
N2	1	N2
N3		
N4		
N5		

N1 → N2 and N2 → N2

1 + 0 = 1

N1 → N2 and N2 → N3

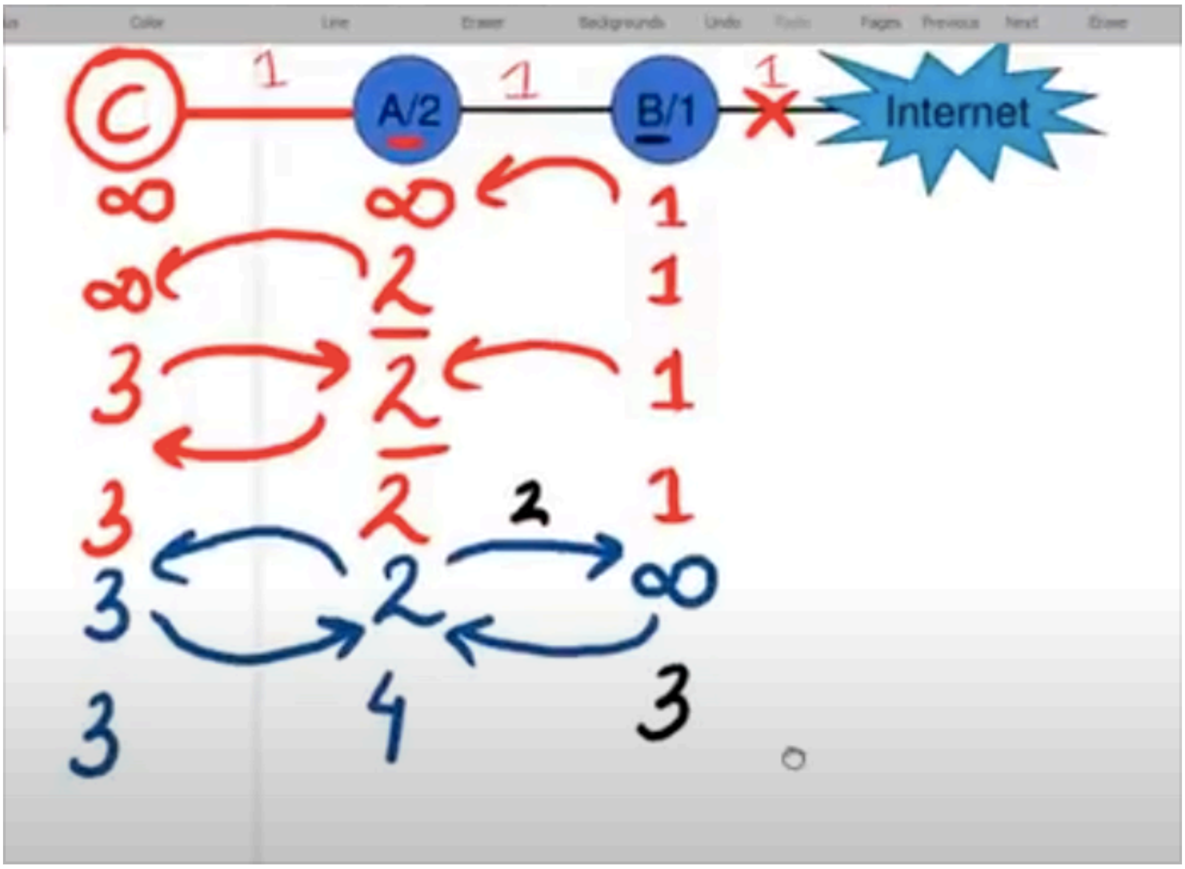
-> this process continue 3-4 times until we got the shortest path in each table

-> and even after the shortest path it will continue to update the table if lets say one path in break or error occurred in any of them it will change the table accordingly with the next path.

-> Note :

Count to infinity problem in distance vector routing,

Here due to break in first link everything got super confused …

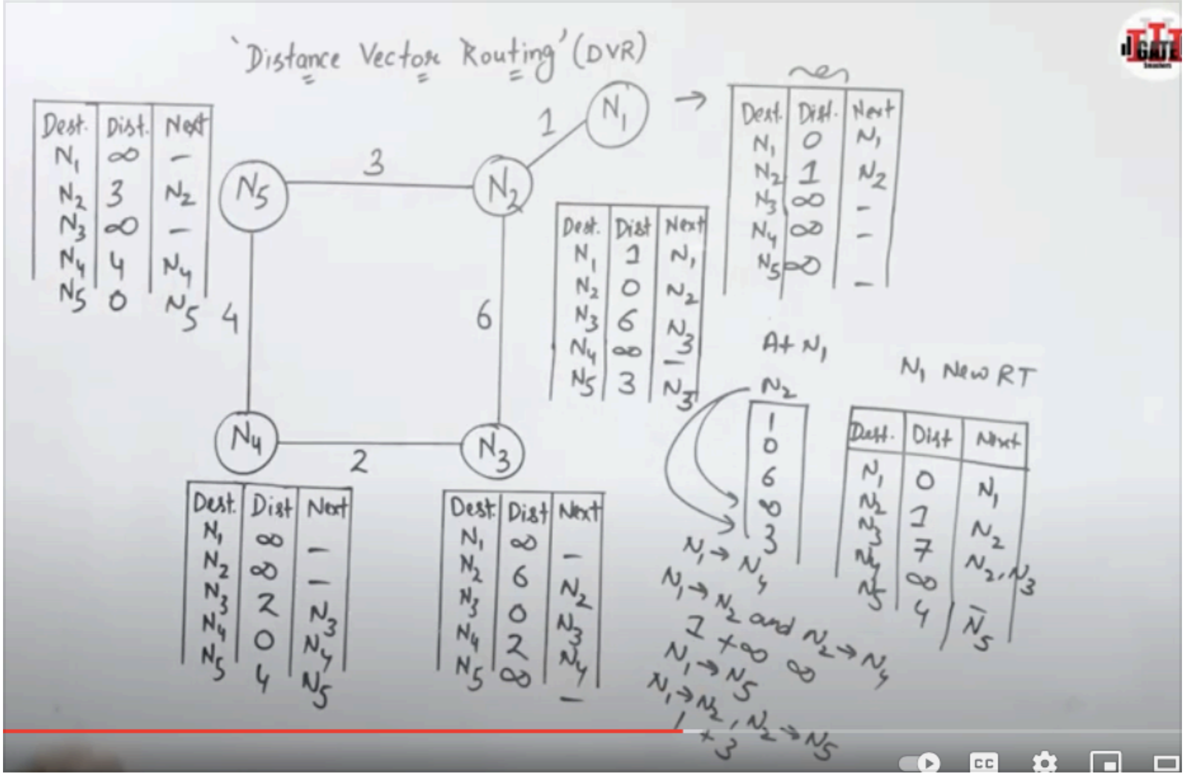


Distance Vector Routing (DVR) Protocol is a method used by routers to find the best path for data to travel across a network. Each router keeps a table that shows the shortest distance to every other router, based on the number of hops (or steps) needed to reach them. Routers share this information with their neighbors, allowing them to update their tables and find the most efficient routes. This protocol helps ensure that data moves quickly and smoothly through the network.

The protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm (or known as the Bellman-Ford algorithm).

Bellman-Ford Basics

Each router maintains a Distance Vector table containing the distance between itself and All possible destination nodes. Distances, based on a chosen metric, are computed using information from the neighbors' distance vectors.



'Distance Vector Routing' (DVR)

Dest.	Dist.	Next
N1	∞	–
N2	3	N2
N3	∞	–
N4	4	N4
N5	0	N5

Dest.	Dist.	Next
N1	0	N1
N2	1	N2
N3	∞	–
N4	∞	–
N5	∞	–

Dest.	Dist	Next
N1	1	N1
N2	0	N2
N3	6	N3
N4	∞	–
N5	3	N5

Dest.	Dist	Next
N1	∞	–
N2	∞	–
N3	2	N3
N4	0	N4
N5	4	N5

Dest.	Dist	Next
N1	∞	–
N2	6	N2
N3	0	N3
N4	2	N4
N5	∞	–

At N1

N2
1
0
6
∞
3

N1 → N4
N1 → N2 and N2 → N4
1 + ∞   ∞
N1 → N5
N1 → N2, N2 → N5
1 + 3

N1 New RT

Dest.	Dist	Next
N1	0	N1
N2	1	N2
N3	7	N2, N3
N4	∞	–
N5	4	N5

How Distance Vector Algorithm works?

- A router transmits its distance vector to each of its neighbors in a routing packet.
- Each router receives and saves the most recently received distance vector from each of its neighbors.
- A router recalculates its distance vector when:
    - It receives a distance vector from a neighbor containing different information than before.
    - It discovers that a link to a neighbor has gone down.

The DV calculation is based on minimizing the cost to each destination

Dx(y) = Estimate of least cost from x to y

C(x,v) = Node x knows cost to each neighbor v

Dx = [Dx(y): y ? N] = Node x maintains distance vector

Node x also maintains its neighbors' distance vectors

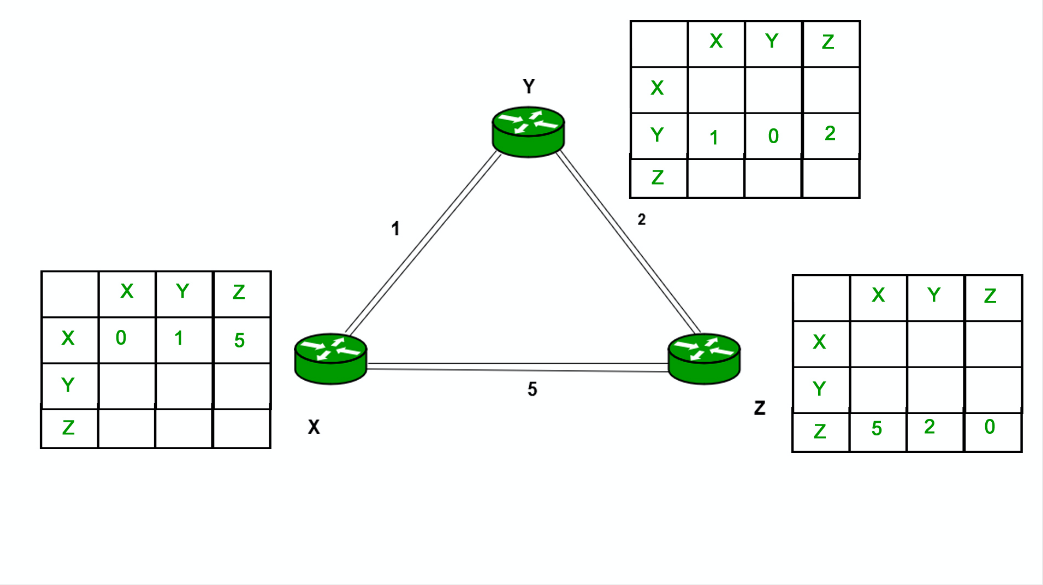
– For each neighbor v, x maintains Dv = [Dv(y): y ? N]

Note:

- From time-to-time, each node sends its own distance vector estimate to neighbors.
- When a node x receives new DV estimate from any neighbor v, it saves v's distance vector and it updates its own DV using B-F equation: Dx(y) = min { C(x,v) + Dv(y), Dx(y) } for each node y ? N

•

Example :
Consider 3-routers X, Y and Z as shown in figure. Each router have their routing table. Every routing table will contain distance to the destination nodes.

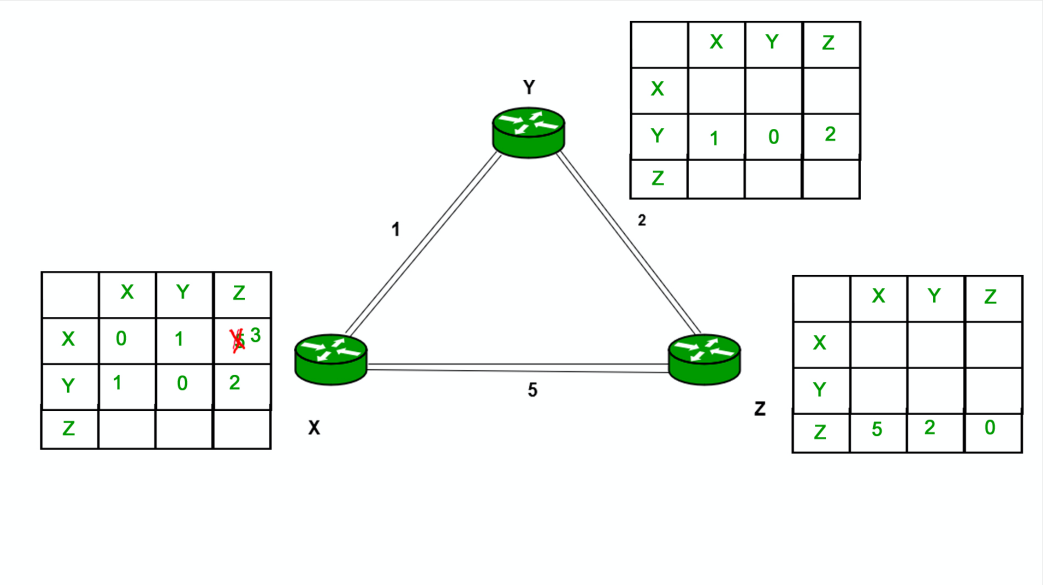


	X	Y	Z
X			
Y	1	0	2
Z			

	X	Y	Z
X	0	1	5
Y			
Z			

	X	Y	Z
X			
Y			
Z	5	2	0

Consider router X , X will share it routing table to neighbors and neighbors will share it routing table to it to X and distance from node X to destination will be calculated using bellmen- ford equation.
 Dx(y) = min { C(x,v) + Dv(y)} for each node y ? N
As we can see that distance will be less going from X to Z when Y is intermediate node(hop) so it will be update in routing table X.

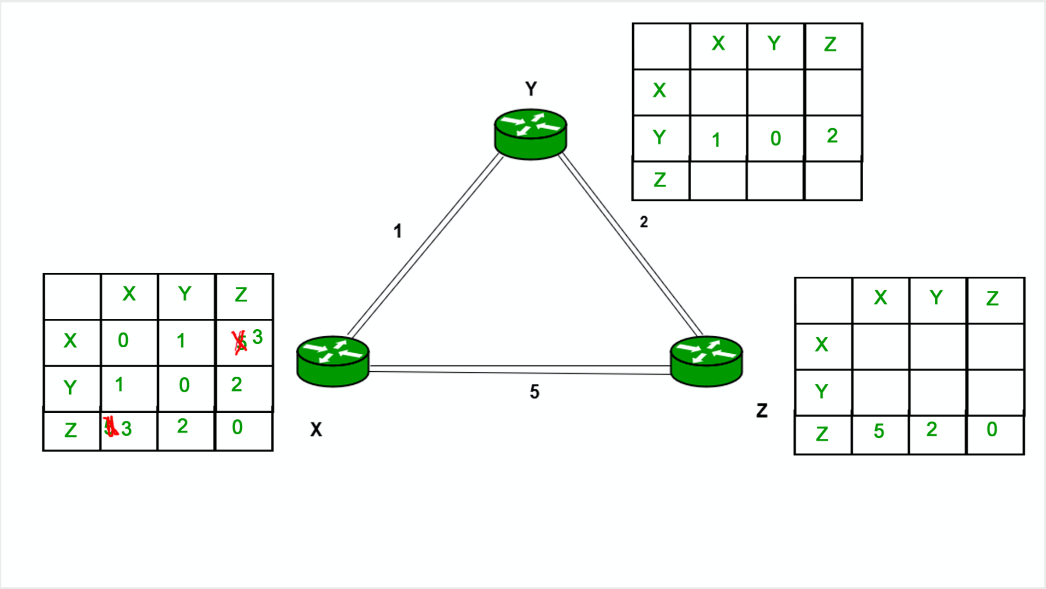


	X	Y	Z
X			
Y	1	0	2
Z			

	X	Y	Z
X	0	1	~~5~~ 3
Y	1	0	2
Z			

	X	Y	Z
X			
Y			
Z	5	2	0

Similarly for Z also –

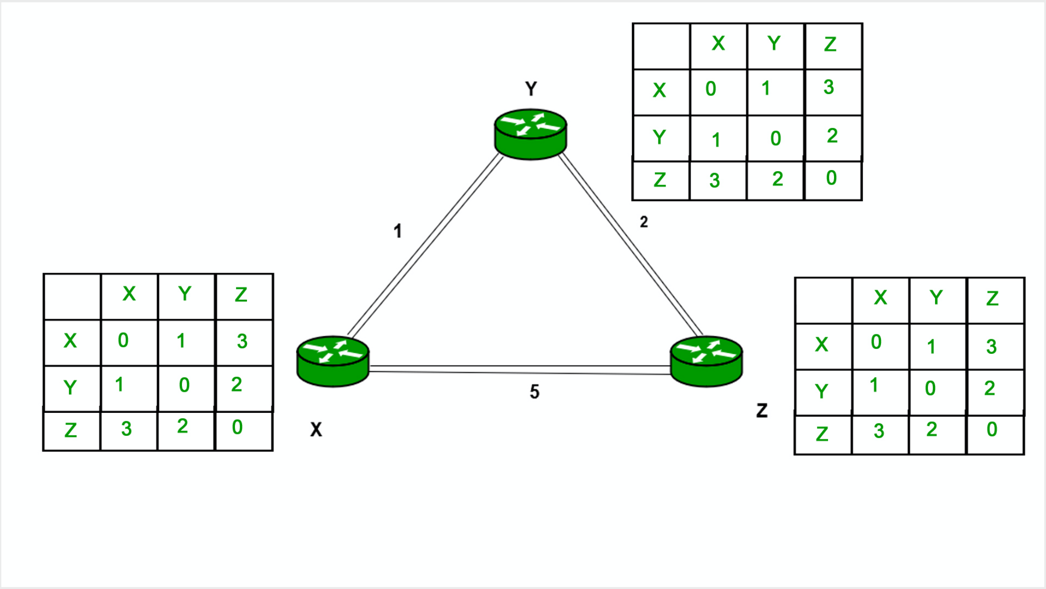


	X	Y	Z
X			
Y	1	0	2
Z			

	X	Y	Z
X	0	1	~~5~~ 3
Y	1	0	2
Z	~~5~~ 3	2	0

	X	Y	Z
X			
Y			
Z	5	2	0

Finally the routing table for all –



	X	Y	Z
X	0	1	3
Y	1	0	2
Z	3	2	0

	X	Y	Z
X	0	1	3
Y	1	0	2
Z	3	2	0

	X	Y	Z
X	0	1	3
Y	1	0	2
Z	3	2	0

Applications of Distance Vector Routing Algorithm

The Distance Vector Routing Algorithm has several uses:

- **Computer Networking**: It helps route data packets in networks.
- **Telephone Systems**: It's used in some telephone switching systems.
- **Military Applications**: It has been used to route missiles.

Advantages of Distance Vector routing

- **Shortest Path**: Distance Vector Routing finds the shortest path for data to travel in a network.
- **Usage**: It is used in local, metropolitan, and wide-area networks.

- **Easy Implementation**: The method is simple to set up and doesn’t require many resources.

Disadvantages of Distance Vector Routing Algorithm

- It is slower to converge than link state.
- It is at risk from the count-to-infinity problem.
- It creates more traffic than link state since a hop count change must be propagated to all routers and processed on each router. Hop count updates take place on a periodic basis, even if there are no changes in the network topology, so bandwidth-wasting broadcasts still occur.
- For larger networks, distance vector routing results in larger routing tables than link state since each router must know about all other routers. This can also lead to congestion on WAN links.

Link State Routing

-> Link state routing maintains table with some additional fields of sequence number number and ttl = time to leave field.
-> ttl field is used to prevent looping of a packet.
-> they shares their neighbors info after initial table to everyone using flooding... while in distance vector we only shares with router neighbors.
-> due to flooding chances of congestion and bandwidth uilization is high
-> but after the flooding each router has information of every other router in the network hence, It can create the same graph internally.
-> now it can use dijistra’s algorithm to calculate the shortest path for each router.

Link state routing is a technique in which each router shares the knowledge of its neighborhood with every other router in the internetwork.

The three keys to understand the Link State Routing algorithm:

- **Knowledge about the neighborhood:** Instead of sending its routing table, a router sends the information about its neighborhood 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 internetwork except its neighbors. This process is known as Flooding. Every router that receives the packet sends the copies to all its neighbors. 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 neighbors.
- **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.

- The Link state routing algorithm is also known as Dijkstra's algorithm which is used to find the shortest path from one node to every other node in the network.
- The Dijkstra's algorithm is an iterative, and it has the property that after k^{th} iteration of the algorithm, the least cost paths are well known for k destination nodes.

Let's describe some notations:

- **c(i , j):** Link cost from node i to node j. If i and j nodes are not directly linked, then c(i , j) = ∞.
- **D(v):** It defines the cost of the path from source code to destination v that has the least cost currently.
- **P(v):** It defines the previous node (neighbor of v) along with current least cost path from source to v.
- **N:** It is the total number of nodes available in the network.

Algorithm

Initialization

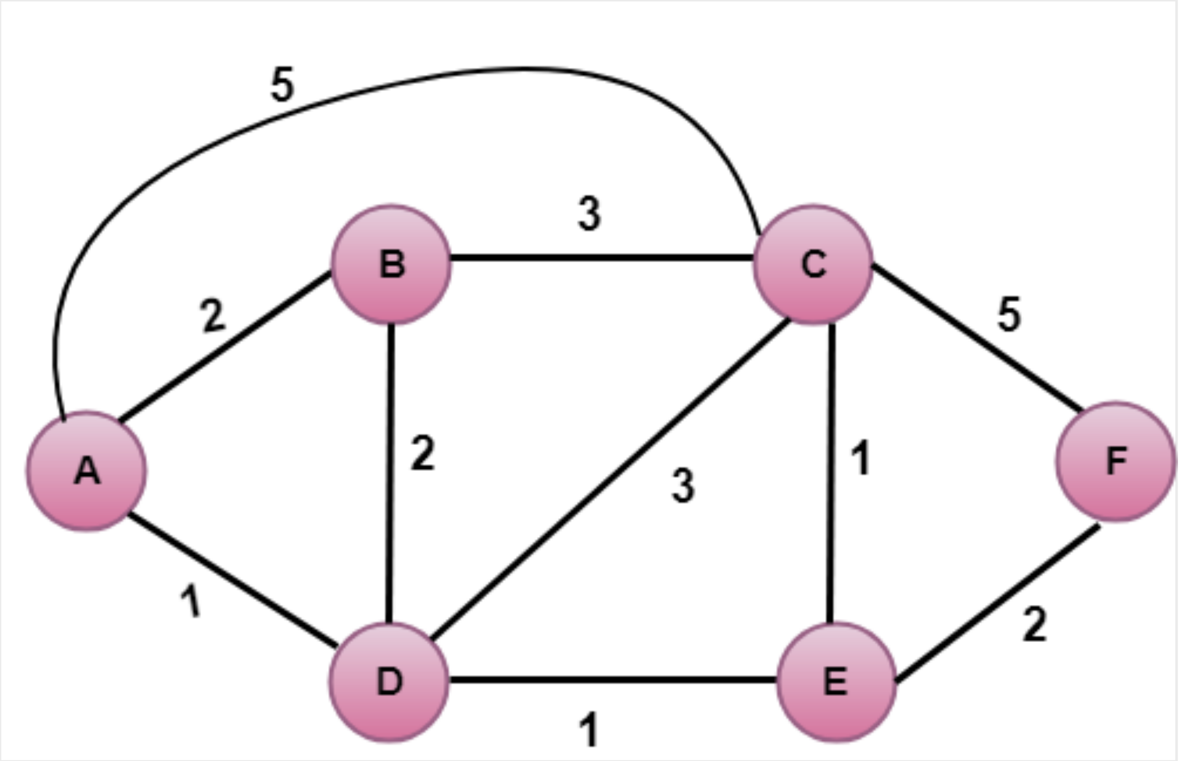
```
N = {A}     // A is a root node.
for all nodes v
if v adjacent to A
then D(v) = c(A,v)
else D(v) = infinity
```

loop

```
find w not in N such that D(w) is a minimum.
Add w 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
```

In the above algorithm, an initialization step is followed by the loop. The number of times the loop is executed is equal to the total number of nodes available in the network.

Let's understand through an example:



In the above figure, source vertex is A.

Step 1:

The first step is an initialization step. The currently known least cost path from A to its directly attached neighbors, B, C, D are 2,5,1 respectively. The cost from A to B is set to 2, from A to D is set to 1 and from A to C is set to 5. The cost from A to E and F are set to infinity as they are not directly linked to A.

Step	N	D(B),P (B)	D(C),P (C)	D(D),P (D)	D(E),P (E)	D(F),P (F)
1	A	2,A	5,A	1,A	∞	∞

Step 2:

In the above table, we observe that vertex D contains the least cost path in step 1. Therefore, it is added in N. Now, we need to determine a least-cost path through D vertex.

a) Calculating shortest path from A to B

1. v = B, w = D
2. D(B) = min(D(B) , D(D) + c(D,B))
3. = min(2, 1+2)>
4. = min(2, 3)
5. The minimum value is 2. Therefore, the currently shortest

path from A to B is 2.

b) Calculating shortest path from A to C

1. v = C, w = D
2. D(B) = min(D(C) , D(D) + c(D,C))
3.   = min(5, 1+3)
4.   = min(5, 4)
5. The minimum value is 4. Therefore, the currently shortest path from A to C is 4.</p>

c) Calculating shortest path from A to E

1. v = E, w = D
2. D(B) = min(D(E) , D(D) + c(D,E))
3.   = min(∞, 1+1)
4.   = min(∞, 2)
5. The minimum value is 2. Therefore, the currently shortest path from A to E is 2.

Note: The vertex D has no direct link to vertex E. Therefore, the value of D(F) is infinity.

Step	N	D(B),P (B)	D(C),P (C)	D(D),P (D)	D(E),P (E)	D(F),P (F)
1	A	2,A	5,A	1,A	∞	∞
2	AD	2,A	4,D		2,D	∞

Step 3:

In the above table, we observe that both E and B have the least cost path in step 2. Let's consider the E vertex. Now, we determine the least cost path of remaining vertices through E.

a) Calculating the shortest path from A to B.

1. v = B, w = E
2. D(B) = min(D(B) , D(E) + c(E,B))
3.   = min(2 , 2+ ∞)
4.   = min(2, ∞)
5. The minimum value is 2. Therefore, the currently shortest path from A to B is 2.

b) Calculating the shortest path from A to C.

1. v = C, w = E
2. D(B) = min(D(C) , D(E) + c(E,C))
3.   = min(4 , 2+1)
4.   = min(4,3)
5. The minimum value is 3. Therefore, the currently shortest path from A to C is 3.

c) Calculating the shortest path from A to F.

1. v = F, w = E
2. D(B) = min(D(F) , D(E) + c(E,F))
3. = min(∞ , 2+2)
4. = min(∞ ,4)
5. The minimum value is 4. Therefore, the currently shortest path from A to F is 4.

Step	N	D(B),P (B)	D(C),P (C)	D(D),P (D)	D(E),P (E)	D(F),P (F)
1	A	2,A	5,A	1,A	∞	∞
2	AD	2,A	4,D		2,D	∞
3	ADE	2,A	3,E			4,E

Step 4:

In the above table, we observe that B vertex has the least cost path in step 3. Therefore, it is added in N. Now, we determine the least cost path of remaining vertices through B.

a) Calculating the shortest path from A to C.

1. v = C, w = B
2. D(B) = min(D(C) , D(B) + c(B,C))
3. = min(3 , 2+3)
4. = min(3,5)
5. The minimum value is 3. Therefore, the currently shortest path from A to C is 3.

b) Calculating the shortest path from A to F.

1. v = F, w = B
2. D(B) = min(D(F) , D(B) + c(B,F))
3. = min(4, ∞)
4. = min(4, ∞)
5. The minimum value is 4. Therefore, the currently shortest path from A to F is 4.

Step	N	D(B),P (B)	D(C),P (C)	D(D),P (D)	D(E),P (E)	D(F),P (F)
1	A	2,A	5,A	1,A	∞	∞
2	AD	2,A	4,D		2,D	∞
3	ADE	2,A	3,E			4,E
4	ADEB		3,E			4,E

Step 5:

In the above table, we observe that C vertex has the least cost path in step 4. Therefore, it is added in N. Now, we determine the least cost path of remaining vertices through C.

a) Calculating the shortest path from A to F.

1. v = F, w = C
2. D(B) = min(D(F) , D(C) + c(C,F))
3. = min(4, 3+5)
4. = min(4,8)
5. The minimum value is 4. Therefore, the currently shortest path from A to F is 4.

Step	**N**	**D(B),P (B)**	**D(C),P (C)**	**D(D),P (D)**	**D(E),P (E)**	**D(F),P (F)**
1	A	2,A	5,A	1,A	∞	∞
2	AD	2,A	4,D		2,D	∞
3	ADE	2,A	3,E			4,E
4	ADEB		3,E			4,E
5	ADEBC					4,E

Final table:

Step	**N**	**D(B),P (B)**	**D(C),P (C)**	**D(D),P (D)**	**D(E),P (E)**	**D(F),P (F)**
1	A	2,A	5,A	1,A	∞	∞
2	AD	2,A	4,D		2,D	∞
3	ADE	2,A	3,E			4,E
4	ADEB		3,E			4,E
5	ADEBC					4,E
6	ADEBCF					

Disadvantage:

Heavy traffic is created in Line state routing due to Flooding. Flooding can cause an infinite looping, this problem can be solved by using Time-to-leave field

What is hierarchical routing?

In hierarchical routing, the routers are divided into regions. Each router has complete details about how to route packets to destinations within its own region. But it does not have any idea about the internal structure of other regions.

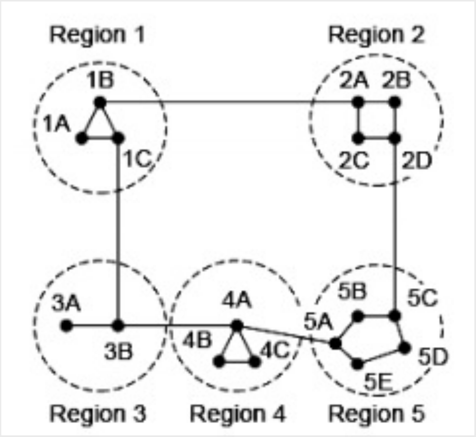
As we know, in both LS and DV algorithms, every router needs to save some information about other routers. When network size is growing, the number of routers in the network will increase. Therefore, the size of routing table increases, then routers cannot handle network traffic as efficiently. To overcome this problem we are using hierarchical routing.

In hierarchical routing, routers are classified in groups called regions. Each router has information about the routers in its own region and it has no information about routers in other regions. So, routers save one record in their table for every other region.

For huge networks, a two-level hierarchy may be insufficient hence, it may be necessary to group the regions into clusters, the clusters into zones, the zones into groups and so on.

Example

Consider an example of two-level hierarchy with five regions as shown in figure –



Let see the full routing table for router 1A which has 17 entries, as shown below –

Full Table for 1A

Dest.	Line	Hops
1A	-	-
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4

4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

When routing is done hierarchically then there will be only 7 entries as shown below –

Hierarchical Table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

Unfortunately, this reduction in table space comes with the increased path length.

Explanation

Step 1 – For example, the best path from 1A to 5C is via region 2, but hierarchical routing of all traffic to region 5 goes via region 3 as it is better for most of the other destinations of region 5.

Step 2 – Consider a subnet of 720 routers. If no hierarchy is used, each router will have 720 entries in its routing table.

Step 3 – Now if the subnet is partitioned into 24 regions of 30 routers each, then each router will require 30 local entries and 23 remote entries for a total of 53 entries.

Example

If the same subnet of 720 routers is partitioned into 8 clusters, each containing 9 regions and each region containing 10 routers. Then what will be the total number of table entries in each router.

Solution

10 local entries + 8 remote regions + 7 clusters = 25 entries.