Network Layer

Network Layer

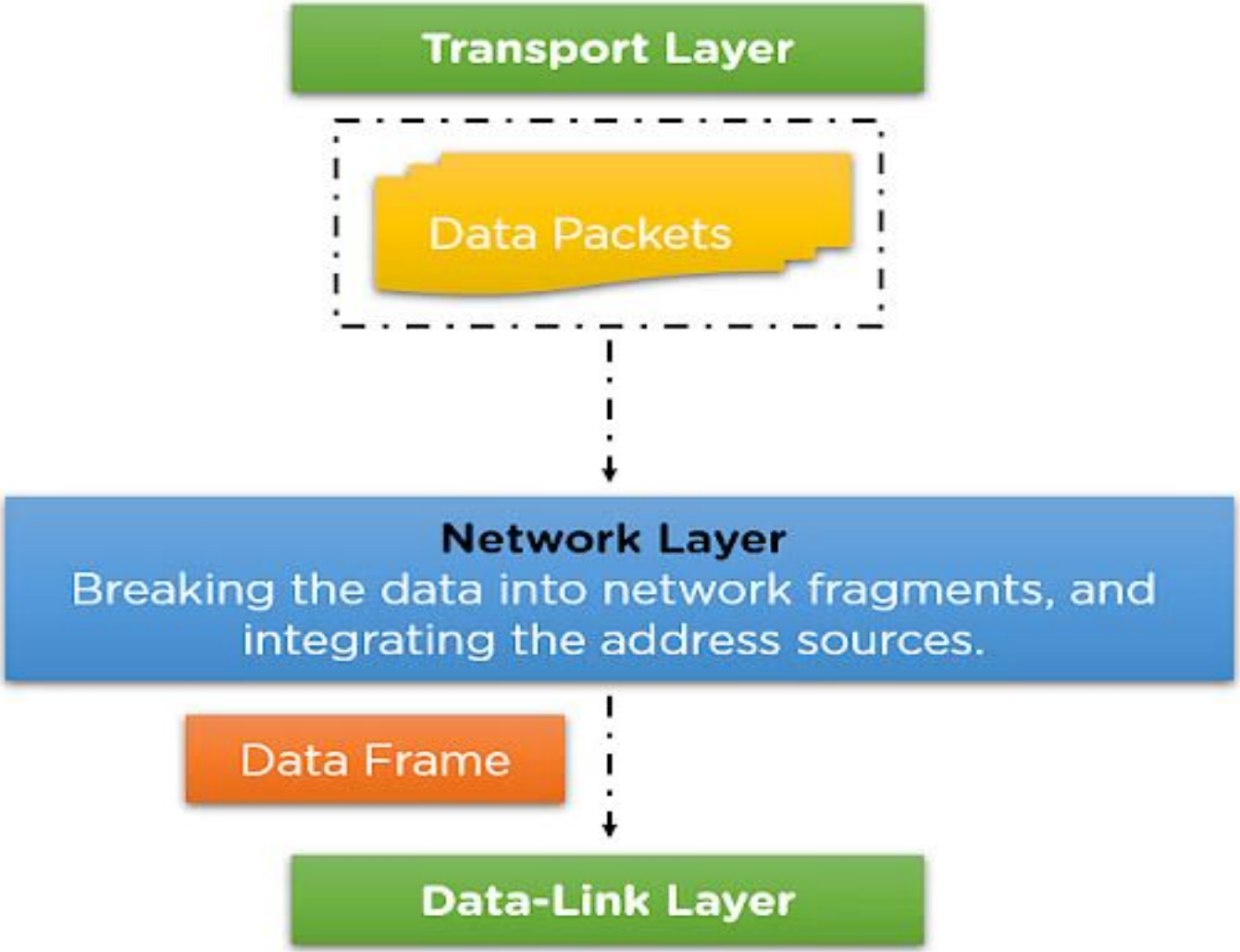
- Network layer
  - design issues - services to the transport layer
  - Routing algorithms-
    - adaptive, non adaptive algorithms,
    - optimality principle,
    - Dijkstra's - shortest path routing algorithm,
    - flow based routing,
    - hierarchical routing,
    - congestion control algorithms – the leaky bucket algorithm, the token bucket algorithm.

Network Layer

Network Layer

The network layer is the third layer in the OSI model and is responsible for **breaking down the data segments into data packet**s and is tasked with **re-assembling** them on the receiver side.

This layer also ensures that the packets are **transmitted over the best possible route** to the destination system, governed by Internet protocols.

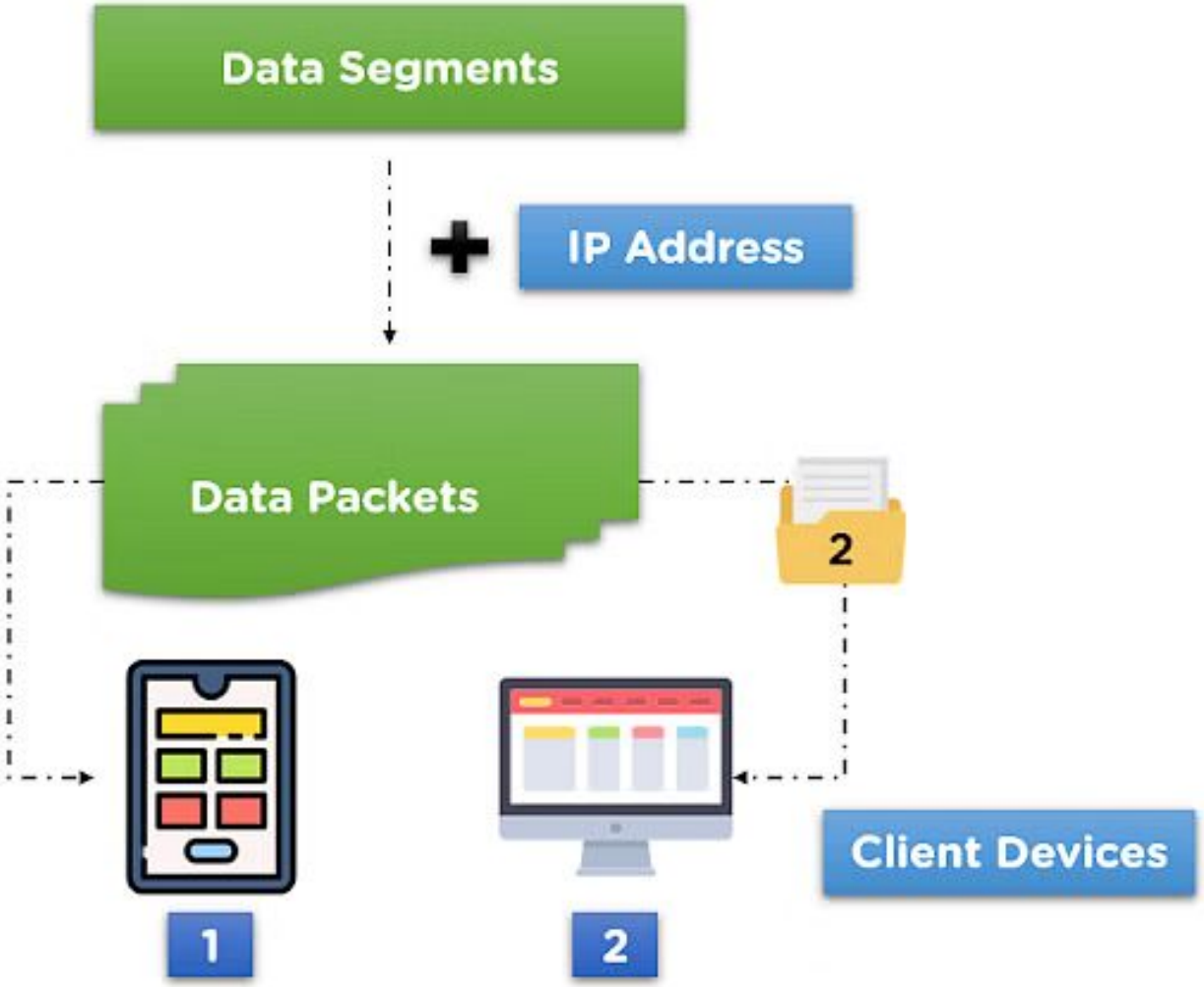

Transport Layer
Data Packets
Network Layer
Breaking the data into network fragments, and integrating the address sources.
Data Frame
Data-Link Layer

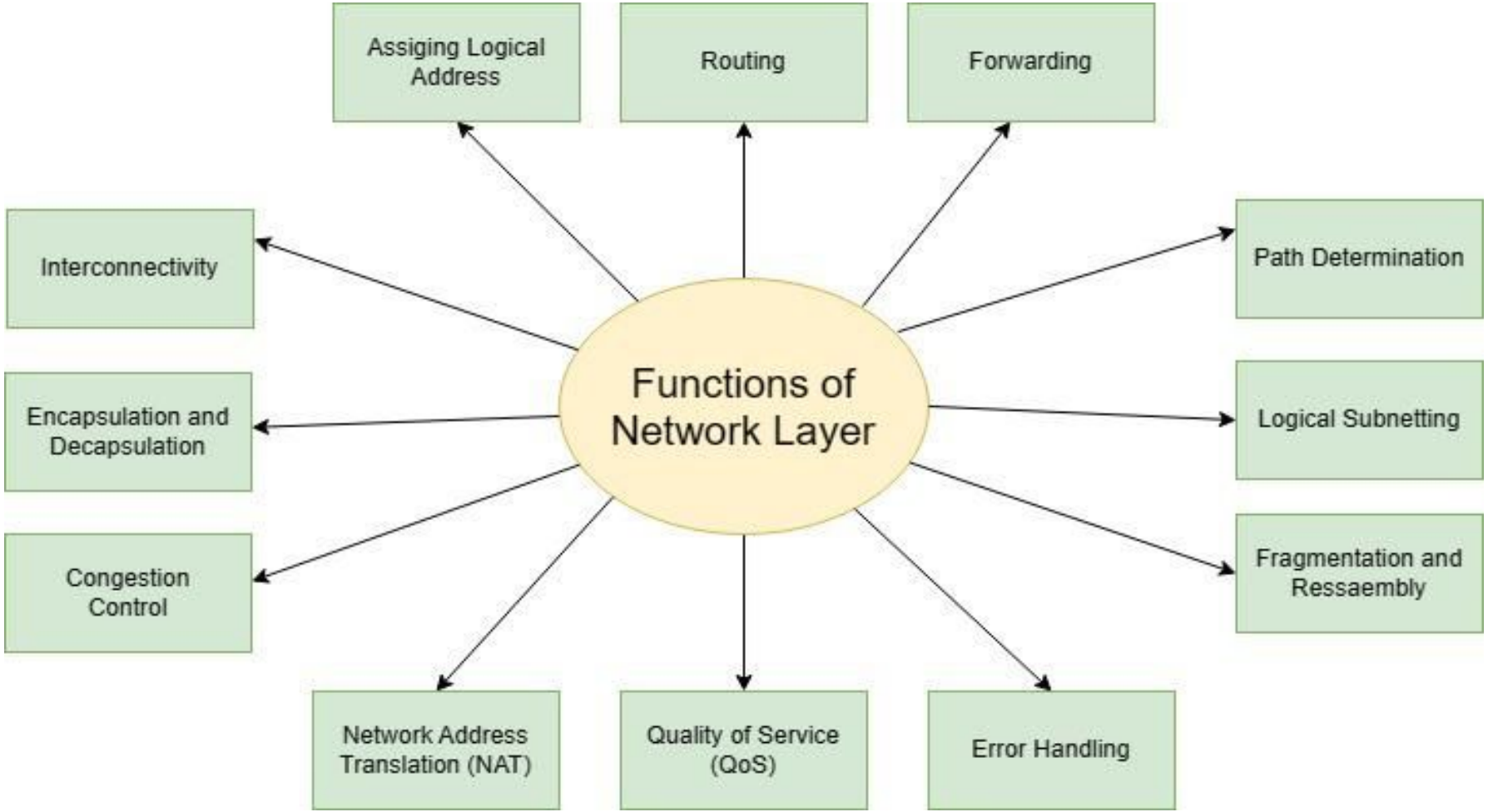
- **The network layer is concerned with getting packets from the source all the way to the destination.**
  - **Getting to the destination may require making many hops at intermediate routers along the way.**

- **The network layer is the lowest layer that deals with end-to-end transmission.**
  - **To achieve its goals, the network layer must know about the topology of the communication subnet (i.e., the set of all routers) and choose appropriate paths through it.**

- **It must also take care to choose routes to avoid overloading some of the communication lines and routers while leaving others idle.**
- **When the source and destination are in different networks, new problems occur.**
  - **It is up to the network layer to deal with them.**

The network layer handles data packets by integrating them with source and destination addresses and also the network protocols

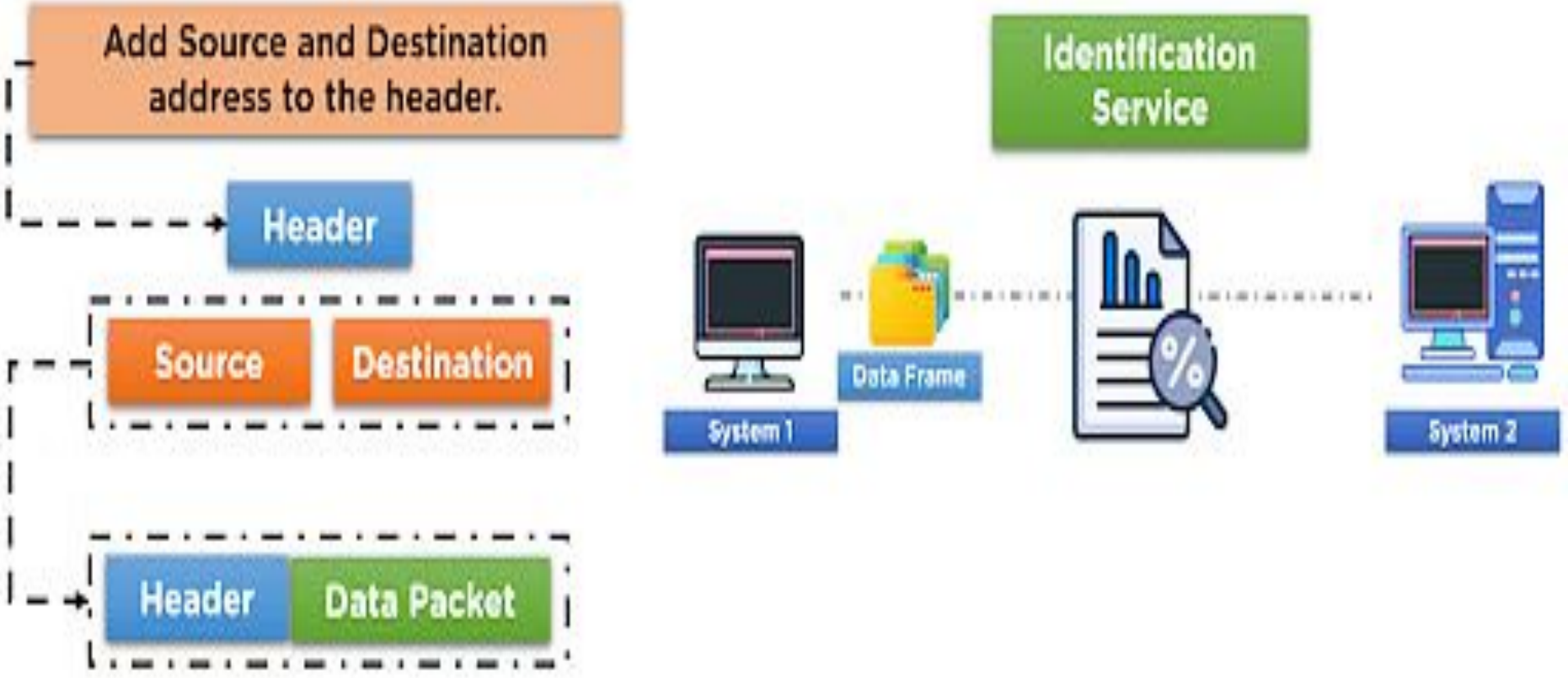

Data Segments
IP Address
Data Packets
2
1
2
Client Devices


Functions of Network Layer
Assiging Logical Address
Routing
Forwarding
Interconnectivity
Path Determination
Encapsulation and Decapsulation
Logical Subnetting
Congestion Control
Fragmentation and Ressaembly
Network Address Translation (NAT)
Quality of Service (QoS)
Error Handling

Functions of the Network Layer

1. Network Addressing

Adding the source and destination address in the header in the network channel. Network addressing is performed to *identify the device*


Add Source and Destination address to the header.
Header
Source
Destination
Header
Data Packet
Identification Service
System 1
Data Frame
System 2

Functions of the Network Layer

2. Inter-Networking

to handle the network connection between multiple devices in the channel.

This task applies multiple protocols available in the network layer

Functions of the Network Layer

3. Packet Routing

Establishing a routing path for the data packet

is responsible for choosing the most suitable routing path


Routing
Router
Sender
Destination

Functions of the Network Layer

4. Packet Handling:

NL's packet handling function is designed to handle the data received from the upper layers of the OSI model. The NL converts the received data into data packets

- **NL is responsible for the following tasks in the communication channel:**
  - **It is responsible for handling the shortest routing path for the data packet in the network channel.**
  - **The network layer converts the received data into data packets for transmission.**
  - **Handles the network layer protocols,**
  - **responsible for maintaining the network traffic in the channel**

Network Layer Design Issues

Network Layer Design Issues

- Some of the issues that the designers of the network layer must address
  - Store-and-Forward Packet Switching
  - Services Provided to the Transport Layer
    - Implementation of Connectionless Service
    - Implementation of Connection-Oriented Service
    - Comparison of Virtual-Circuit and Datagram Networks

Network Layer Design Issues

- ***Store-and-Forward Packet Switching***

Store-and-Forward Packet Switching

- How the network layer protocols operate is given in the figure below

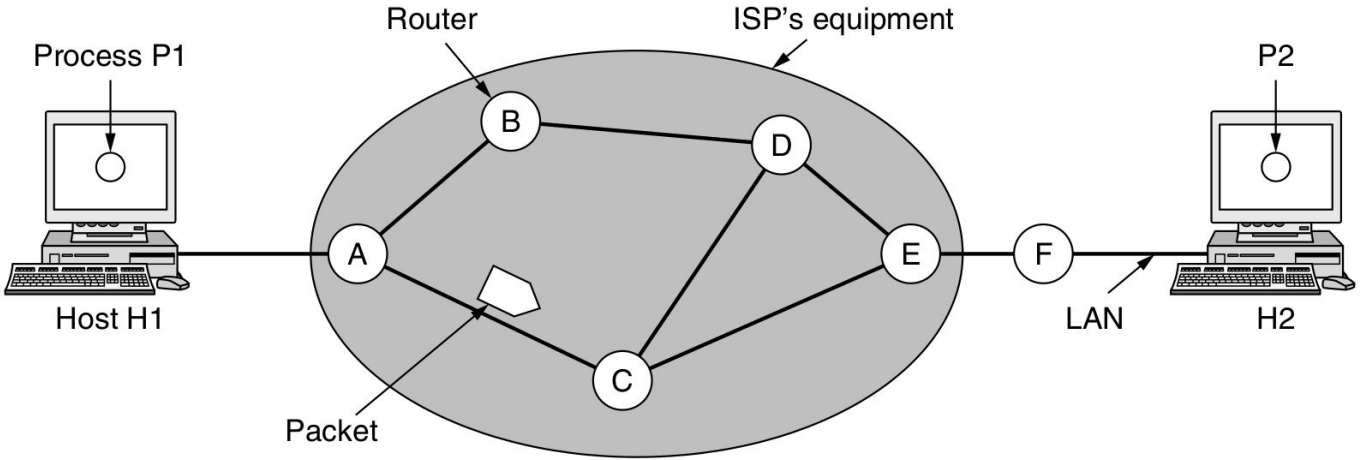


Figure 5-1. The environment of the network layer protocols.

- major components of the network
  - the ISP's equipments - (routers connected by transmission lines), shown inside the shaded oval,
  - and the customers' equipment, shown outside the oval.
- Host H1 is directly connected to one of the ISP's routers, A, perhaps as a home computer that is plugged into a DSL modem.

- In contrast, H2 is on a LAN, which might be an office Ethernet, with a router, F, owned and operated by the customer.
  - This router has a leased line to the ISP's equipment.
- however, routers on customer premises are considered part of the ISP network because they

- A host *transmits* a packet to the nearest router,
  - either on its own LAN
  - or over a point-to-point link to the ISP.
- The packet is stored there until it has fully arrived and the link has finished its processing by verifying the checksum.
- Then it is forwarded to the next router along the path until it reaches the destination host, where it is delivered.
  - This mechanism is store-and-forward packet switching

Network Layer Design Issues

- ***Services Provided by the Network Layer***

- **The services provided by the NL**
  - **Logical Addressing –**

    Network layer adds header to incoming packet which includes *logical address* to identify sender and receiver.
  - **Routing –**

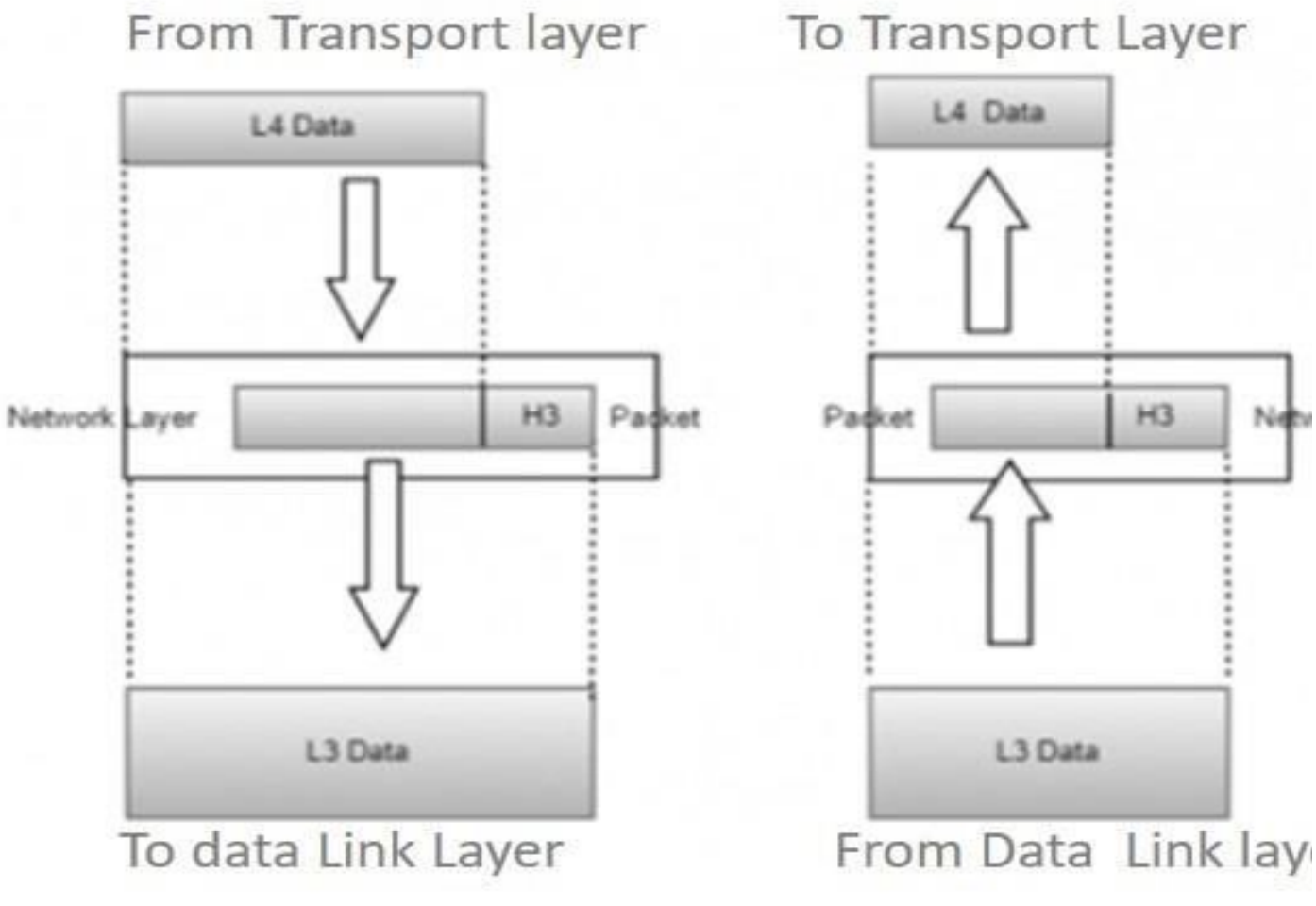
    It is the mechanism provided by Network Layer for *routing the packets to the final destination* in the fastest possible and efficient way.

- **Flow control –**

  This layer routes the packet to another way, If too many packets are present at the same time **preventing bottlenecks and congestion.**

- **Breaks Large Packets –**

  Breaks larger packets into small packets.


From Transport layer
L4 Data
Network Layer
H3
Packet
L3 Data
To data Link Layer
To Transport Layer
L4 Data
Packet
H3
L3 Data
From Data Link lay

Network Layer Design Issues

- ***Services Provided to the Transport Layer***

➢ *Services Provided to the Transport Layer*

- **The services the network layer provides, need to be designed with the following goals in mind**
  - **The services should be *independent* of the router technology.**
  - **The transport layer should be *shielded* from the number, type, and topology of the routers present.**
  - **The network addresses made available to the transport layer should use a *uniform* numbering plan, even across LANs and WANs.**

- **Whether network layer provide**
  - **Connection-oriented service**
  - **Connectionless service**

❖ Internet community opinion

- ➢ Routers move the packets and Subnet is unreliable
- ➢ Host should do error control and flow control
  - ■ These viewpoints lead to the ***connectionless service***
    - Primitives used are
      - ◆ SEND_ PACKET & RECEIVE_PACKET
    - Each packet must contain sender & destination address
    - In particular, **no packet ordering and flow control** should be done because the hosts are going to do that anyway
    - This is an example of the **end-to-end** argument, a design principle that has been very influential in shaping the Internet

Telephone company's opinion

- **The subnet should provide a reliable and connection oriented service**
- **Quality of service is dominant here.**
- **So connection oriented service is to be used**, where a ***communication session is established*** before any useful data can be transferred and where a stream of data is delivered in the same order as it was sent.

❖ A Virtual Circuit –

Transporting data over a packet switched computer network in such a way that it appears as though there is a dedicated physical layer link between the source and destination

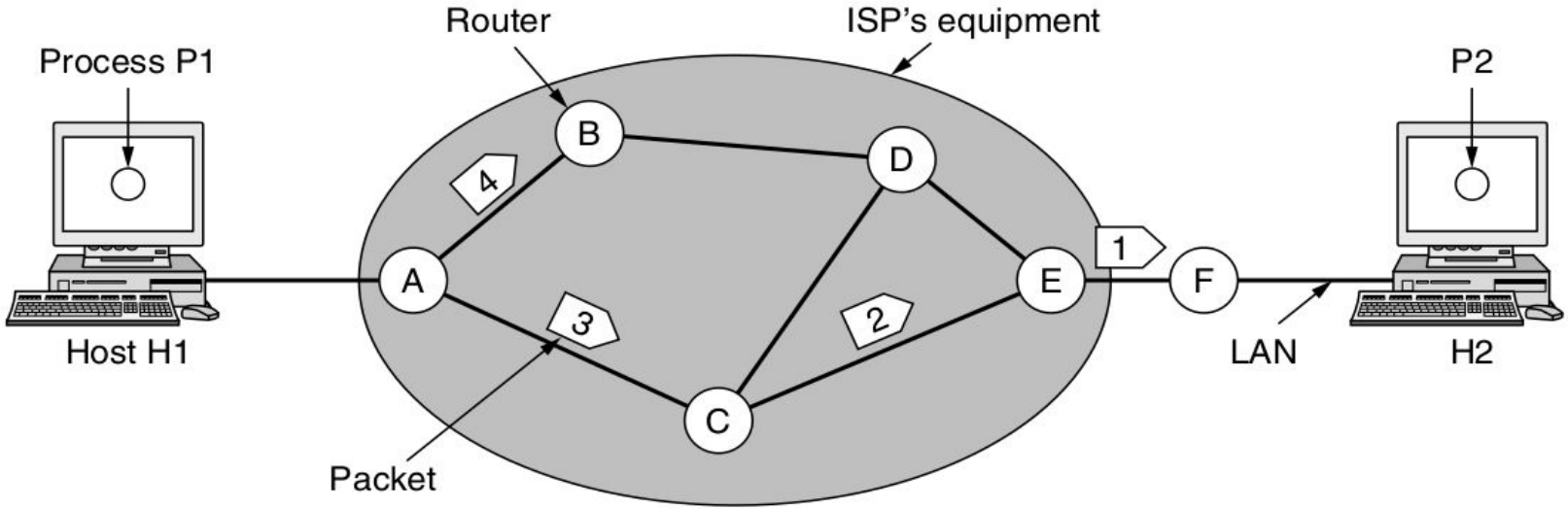
➢ **This connection (communication session) is called a VC (virtual circuit) and the network is called a virtual-circuit network**

- **The internet offers ——--- Connectionless service**
- **The ATM network offers ——----- Connection oriented service**

❖ *Implementation of Connectionless Service*

❖ *Implementation of Connectionless Service*

- ➢ **In connectionless service, the packets are injected into the network individually and routed independently of each other.**
- ➢ **No advance setup is needed.**
- ➢ **In this context, the packets are frequently called datagrams (in analogy with telegrams), and the network is called a datagram network.**



A's table (initially)

Dest.	Line
A	–
B	B
C	C
D	B
E	C
F	C

A's table (later)

A	–
B	B
C	C
D	B
E	B
F	B

C's table

A	A
B	A
C	–
D	D
E	E
F	E

E's table

A	C
B	D
C	C
D	D
E	–
F	F

Figure 5-2. Routing within a datagram network.

- **Suppose that the process P1 sends the message to the transport layer with instructions to deliver it to process P2 on host H2.**
- **The transport layer code runs on H1, typically within the operating system**

- At A, packets 1, 2, and 3 are stored briefly, having arrived on the incoming link and had their checksums verified.
- Then each packet is forwarded according to A's table, onto the outgoing link to C within a new frame.
- Packet 1 is then forwarded to E and then to F.
- When it gets to F, it is sent within a frame over the LAN to H2. Packets 2 and 3 follow the same route.

- **However, something different happens to packet 4.**
- **When it gets to A, it is sent to router B, even though it is also destined for F.**
- **For some reason, A decided to send packet 4 via a different route than that of the first three packets.**
- **Perhaps it has learnt of a traffic jam somewhere along the ACE path and updated its routing table**

- **Every router has an internal table**
  - **Each table entry is a pair consisting of a destination and the outgoing line to use for that destination**
- **Only directly connected lines can be used**

- **The algorithm that manages the tables and makes the routing decisions is called the routing algorithm**

- ***Implementation of Connection-oriented Service***

❖ *Implementation of Connection-oriented Service*

- ➢ **If connection-oriented service is used,**
  - **a *path* from the source router to the destination router must be established before any data packets can be sent.**
  - **This connection is called a VC (virtual circuit) and the network is called a virtual-circuit network.**
  - **virtual circuits is to avoid having to choose a new route for every packet sent**

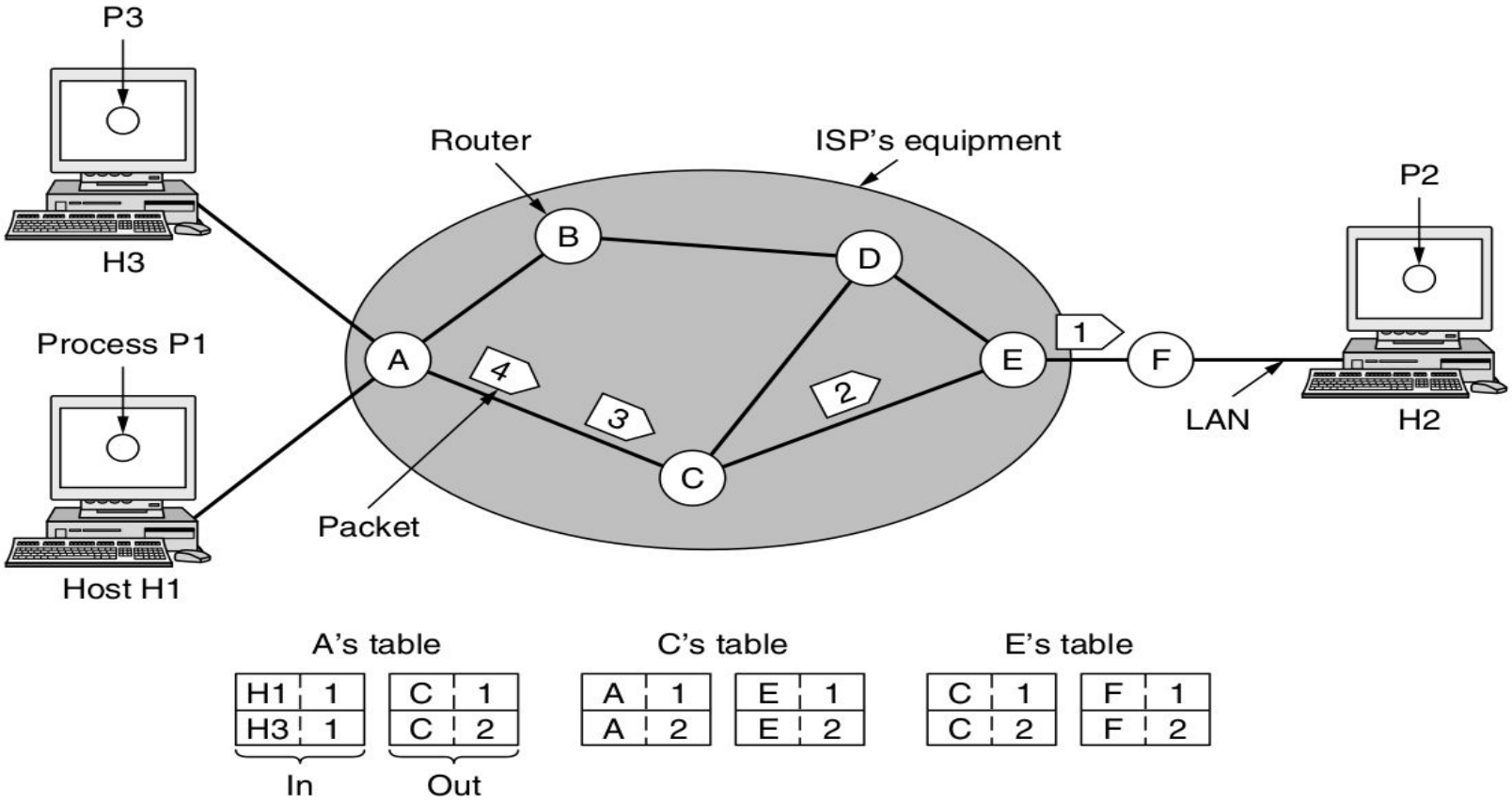


Figure 5-3. Routing within a virtual-circuit network.

- **With connection-oriented service, each packet carries an *identifier* telling which virtual circuit it belongs to**
- **Host H1 has established connection 1 with host H2.**
  - **This connection is remembered as the first entry in each of the routing tables.**

- **The first line of A's table says that if a packet bearing *connection identifier 1* comes in from H1, it is to be sent to router C and given connection identifier 1.**
- **Similarly, the first entry at C routes the packet to E, also with connection identifier 1.**

- if H3 also wants to establish a connection to H2.
- It chooses connection identifier 1 (because it is initiating the connection) and tells the network to establish the virtual circuit.
- This leads to the second row in the tables.

- We have a conflict here because although A can easily distinguish connection 1 packets from H1 from connection 1 packets from H3, C cannot do this.
- For this reason, A assigns a different connection identifier to the outgoing traffic for the second connection
- In some contexts, *this process is called label switching.*
- An example of a connection-oriented network service is MPLS (Multi Protocol Label Switching).

Comparison of Virtual-Circuit and Datagram Networks

Issue	Datagram network	Virtual-circuit network
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Routers do not hold state information about connections	Each VC requires router table space per connection
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow it
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Quality of service	Difficult	Easy if enough resources can be allocated in advance for each VC
Congestion control	Difficult	Easy if enough resources can be allocated in advance for each VC

ROUTING ALGORITHMS

- **The routing algorithm in the network layer software responsible for deciding to which output line an incoming packet should be transmitted on.**
- **If the network uses datagrams internally, this decision must be made a *new for every arriving data packet***

- If the network uses virtual circuits internally, *routing decisions are made only when a new virtual circuit* is being set up. Thereafter, data packets just follow the already established route.
  - This is sometimes called *session routing* because a route remains in force for an entire session

- **Router has two processes inside it.**
  - **One of them handles each packet as it arrives, looking up the outgoing line to use for it in the routing tables.**
    - **This process is forwarding.**
  - **The other process is responsible for filling in and updating the routing tables done by the routing algorithm**

Types of Routing Algorithms

- Routing algorithms can be grouped into two major classes:
  - Non-adaptive
  - Adaptive

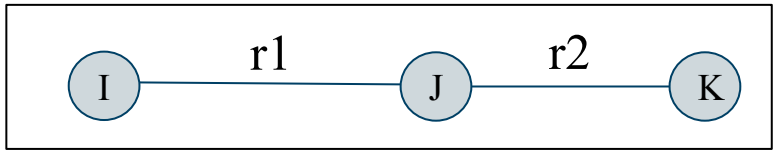
- **Non-adaptive algorithms do *not base their routing decisions* on any measurements or estimates of the current topology and traffic.**
- **Instead, the choice of the route to use to get from I to J (for all I and J) is computed in advance, off- line, and downloaded to the routers when the network is booted.**
- **This procedure is sometimes called static routing.**

- **Adaptive algorithms, in contrast,**
  - **change their routing decisions to reflect changes in the topology, and sometimes changes in the traffic as well.**
  - **These dynamic routing algorithms differ in**
    - **where they get their information (e.g., locally, from adjacent routers, or from all routers),**
    - **when they change the routes (e.g., when the topology changes, or every ΔT seconds as the load changes),**
    - **and what metric is used for optimization**

The Optimality Principle

Optimal routes without regard to network topology or traffic is known as the *optimality principle* (Bellman, 1957).

It states that if router J is on the optimal path from router I to router K, then the optimal path from J to K also falls along the same route.



Eg.,

the part of the route from I to J —- r1 & the rest of the route —--r2 .

If a route better than r2 existed from J to K, it could be concatenated with r1 to improve the route from I to K, contradicting the statement that r1 r2 is optimal.

The set of optimal routes from all sources to a given destination form a tree rooted at the destination.

Such a tree is called a sink tree and is illustrated in Fig

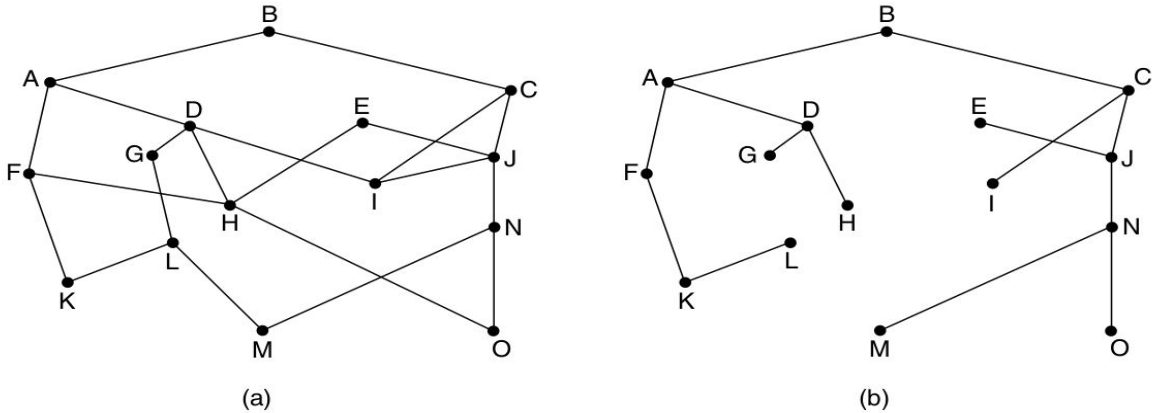


Figure 5-6. (a) A network. (b) A sink tree for router *B*.

where the distance metric is the number of hops. Sink tree is not necessarily unique, other trees with same path lengths may exists.

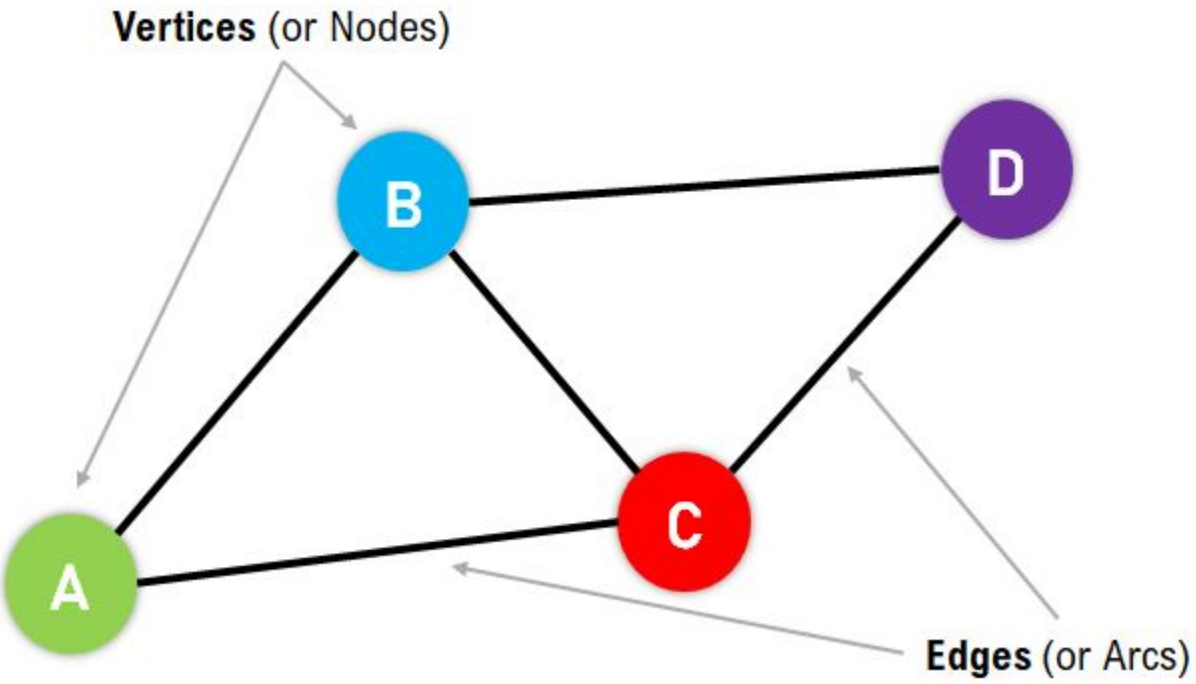
- **The goal of all routing algorithms is to discover and use the sink trees for all routers.**
- **Since a sink tree is indeed a tree, it does not contain any loops, so each packet will be delivered within a finite and bounded number of hops.**

Shortest Path Algorithm

Shortest Path Algorithm

- Build a graph of the network,
  - with each node of the graph representing a router
  - and each edge of the graph representing a communication line, or link.

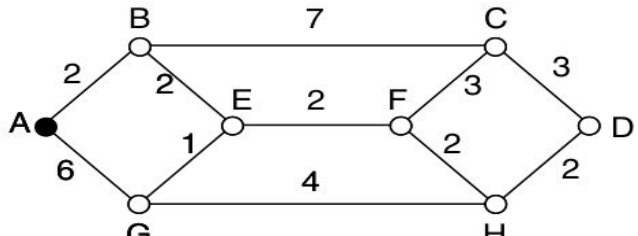
Shortest Path Algorithm



Shortest Path Algorithm

- **To choose a route between a given pair of routers,**
  - **the algorithm just finds the shortest path between them on the graph.**

- **One way of measuring path length**
  - **is the number of hops. Using this metric, the paths ABC and ABE are equally long.**
  - **Another metric is the geographic distance in kilometers, in which case ABC is clearly much longer than ABE**

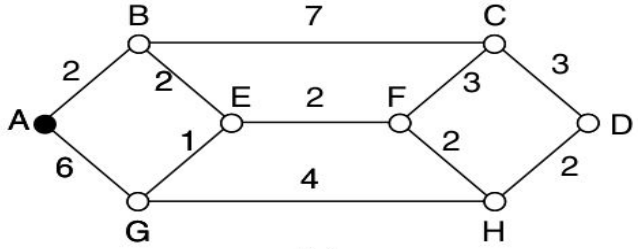


- **Other ways of measuring path length**
  - **labels on the edges could be computed as a function of the distance, bandwidth, average traffic, communication cost, measured delay, and other factors**

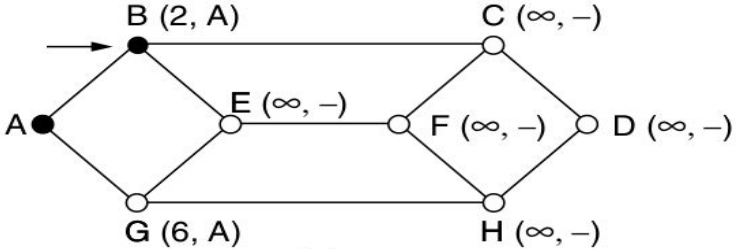
Dijkstra's Shortest Path Algorithm

- **Each node is labeled with its distance from the source node along the best known path.**
- **Initially, no paths are known, so all nodes are labeled with infinity.**
- **As the algorithm proceeds and paths are found, the labels may change, reflecting better paths**
- **A label may be either tentative or permanent (unvisited or visited). Initially, all labels are tentative.**
- **In the given graph (next page..) -**
  - **We want to find the shortest path from A to D**

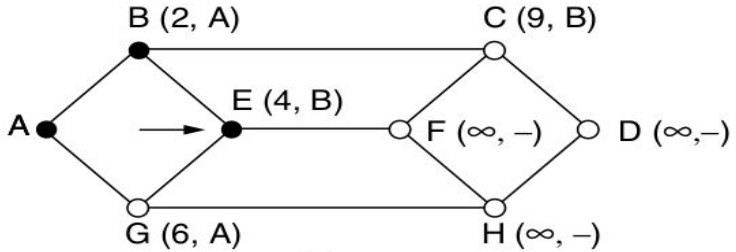
The first six steps used in computing the shortest path from A to D. The arrows indicate the working node.



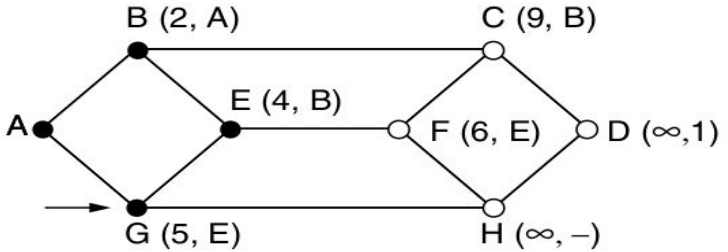
(a)



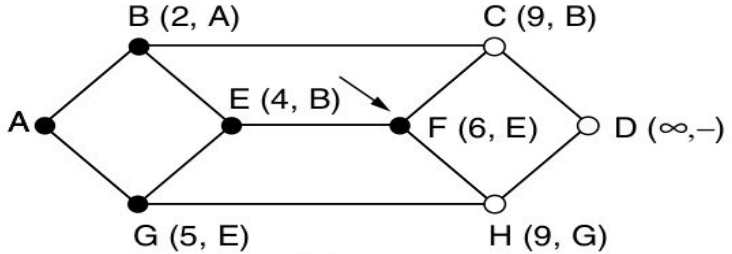
(b)



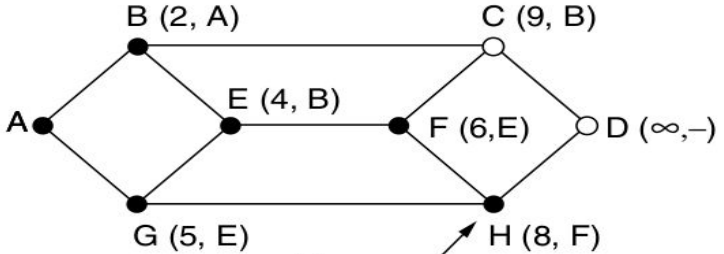
(c)



(d)

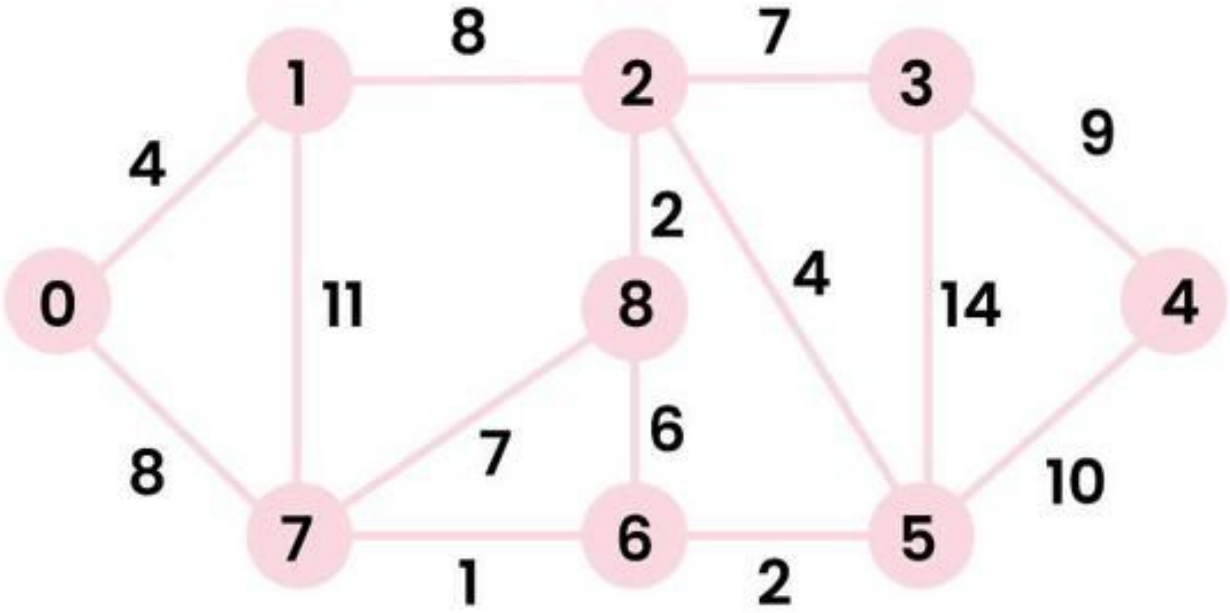


(e)

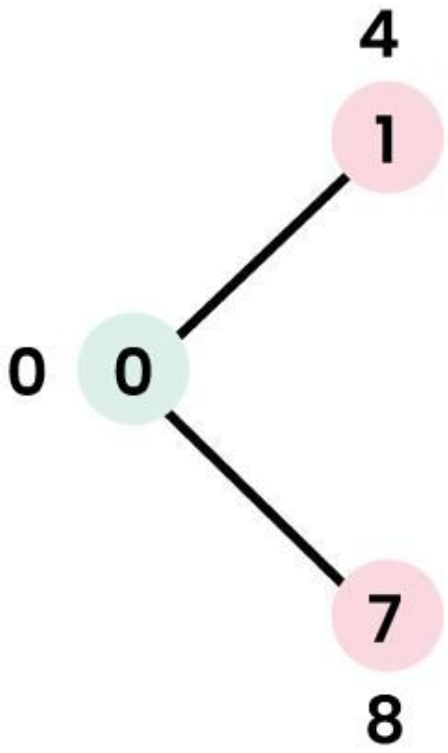


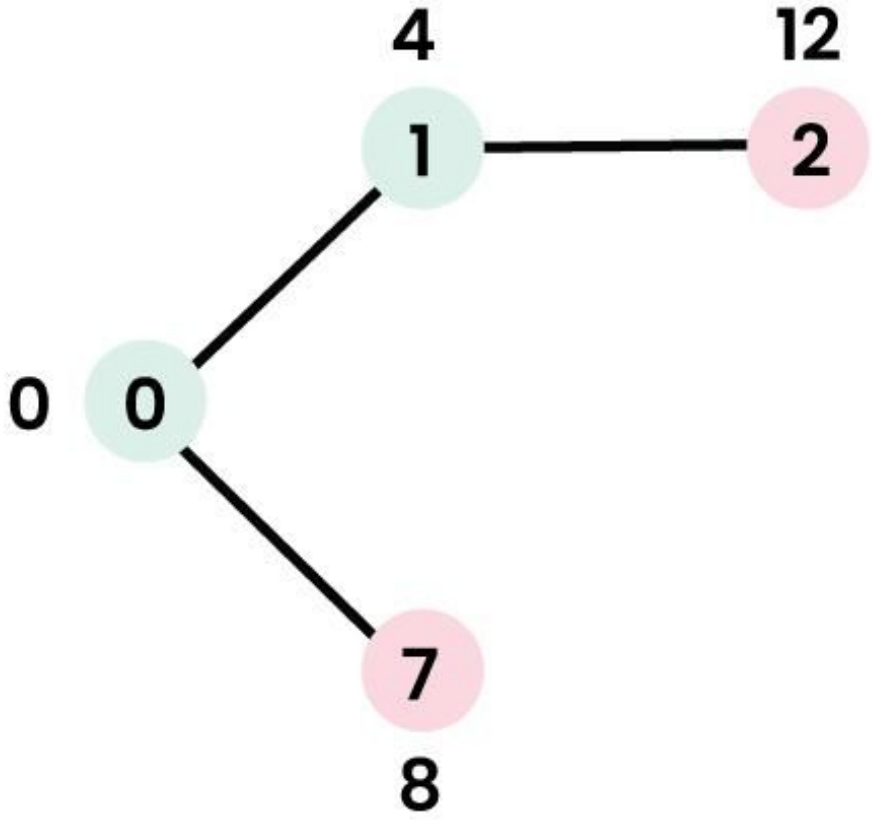
(f)

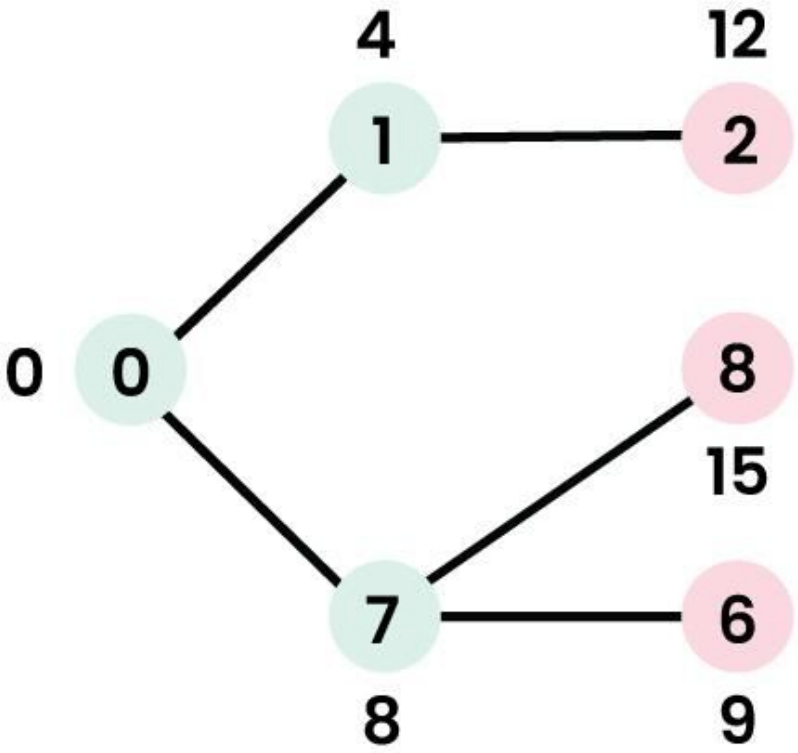
An Example

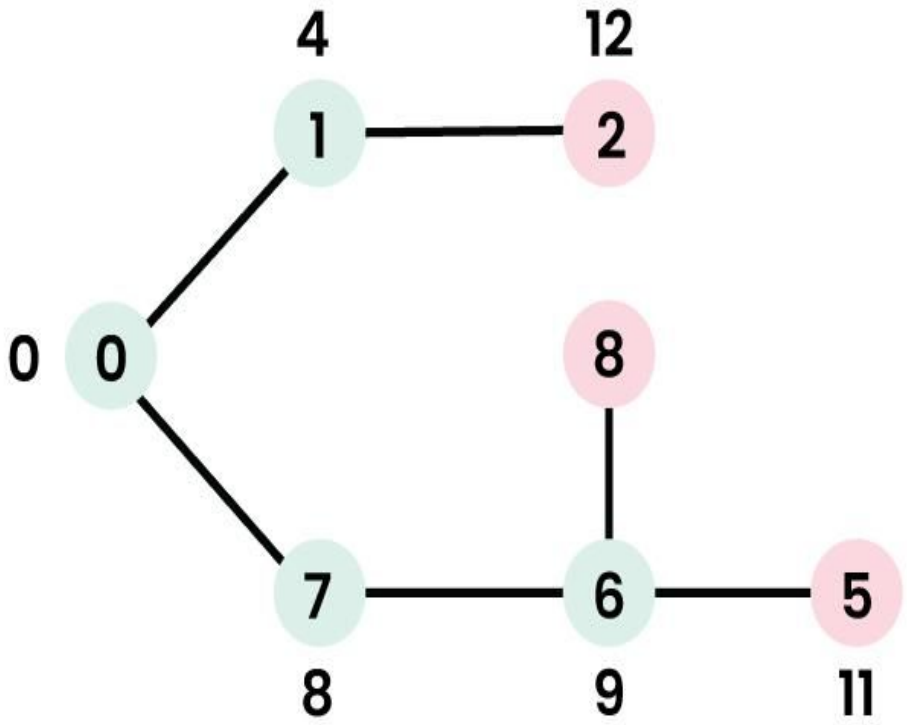


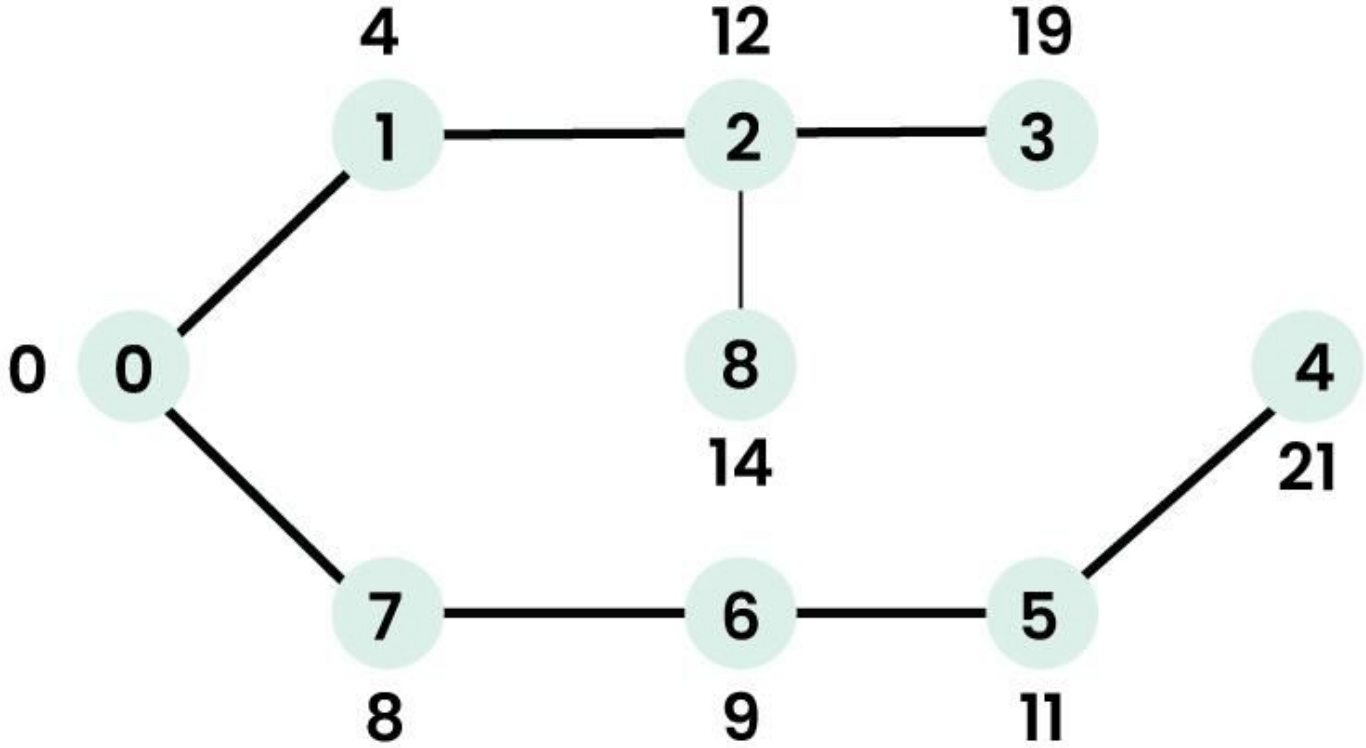
Working of Dijkstra's Algorithm


4
1
0
0
7
8

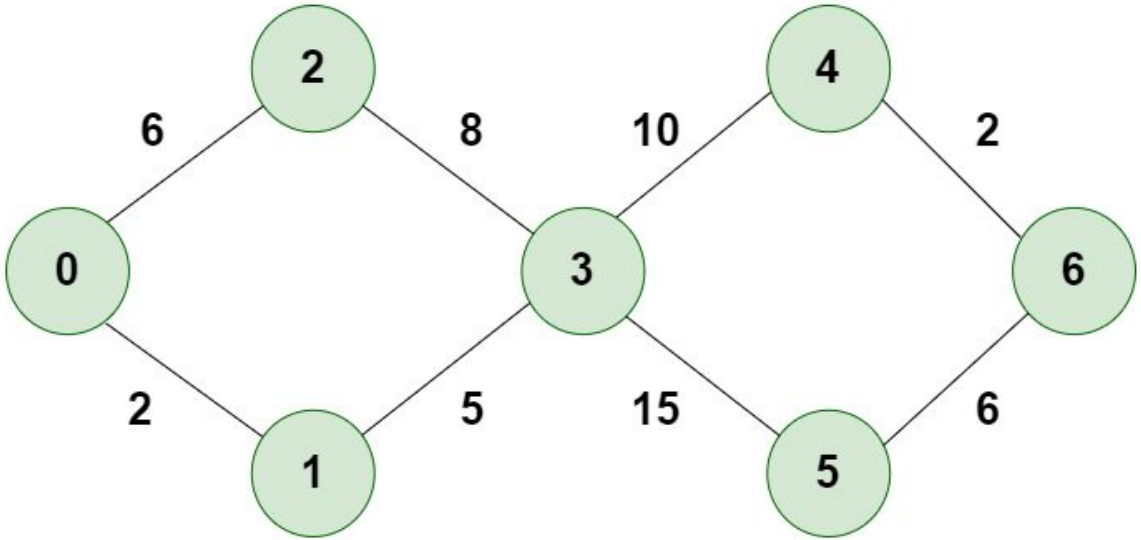

4
12
1
2
0
0
7
8


4
12
1
2
0
0
8
15
7
6
8
9


4
12
1
2
0
0
8
7
6
5
8
9
11


4
12
19
1
2
3
0
0
8
4
14
21
7
6
5
8
9
11

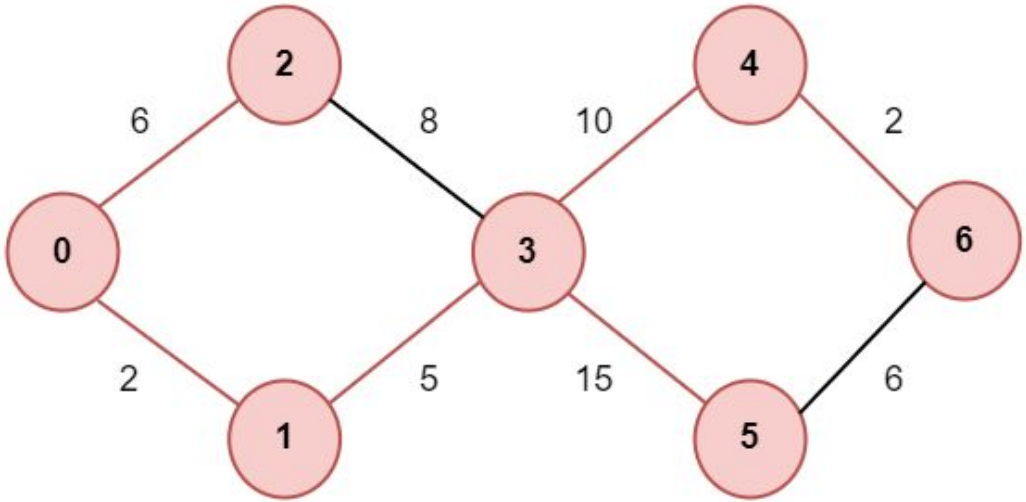
Example Graph



Dijkstra's Algorithm

STEP 5

Mark Node 6 as Visited and add the Distance



Unvisited Nodes

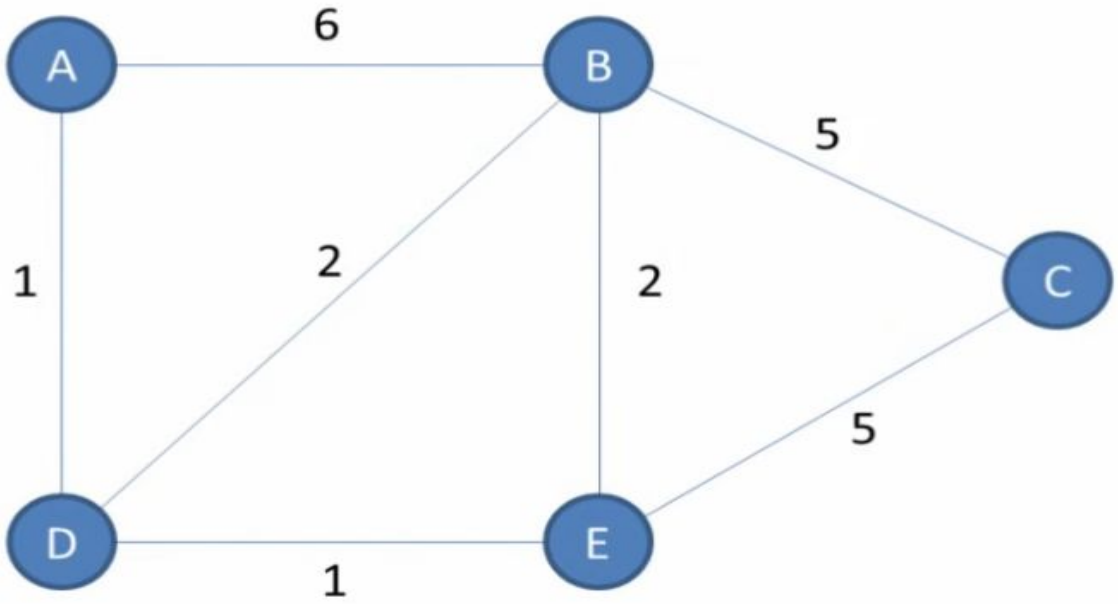
{0,1,2,3,4,5,6}

Distance:

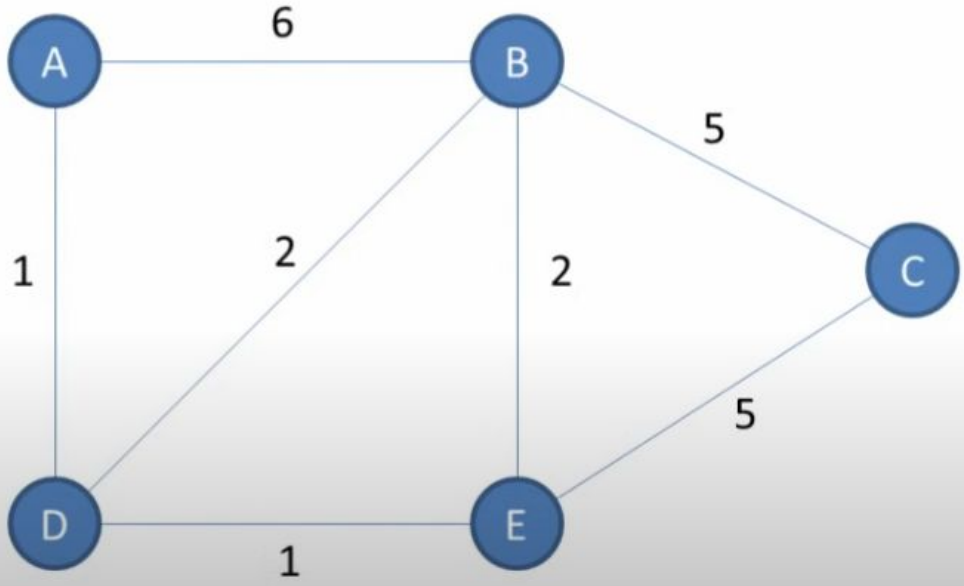
- 0: 0 ✓
- 1: 2 ✓
- 2: 6 ✓
- 3: 7 ✓
- 4: 17 ✓
- 5: 22 ✓
- 6: 19 ✓

Dijkstra's Algorithm

Find the shortest path from vertex A to every other vertex

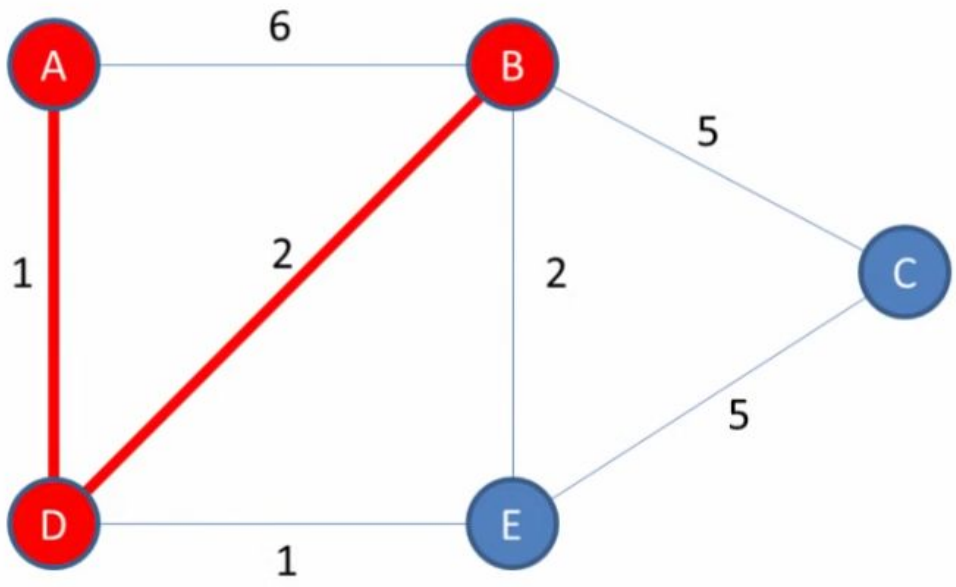


We have the shortest distance from vertex A to every other vertex



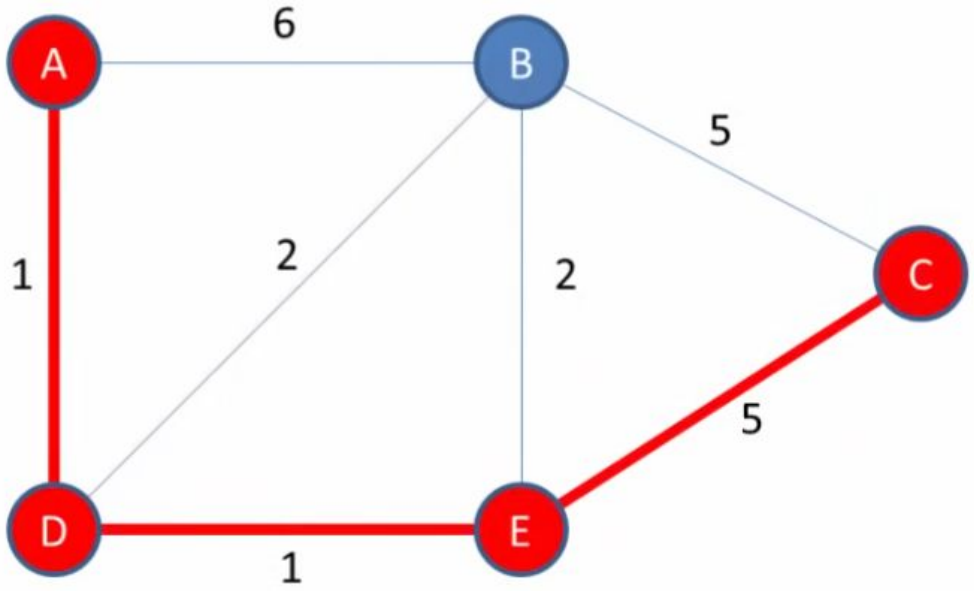
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

A to B



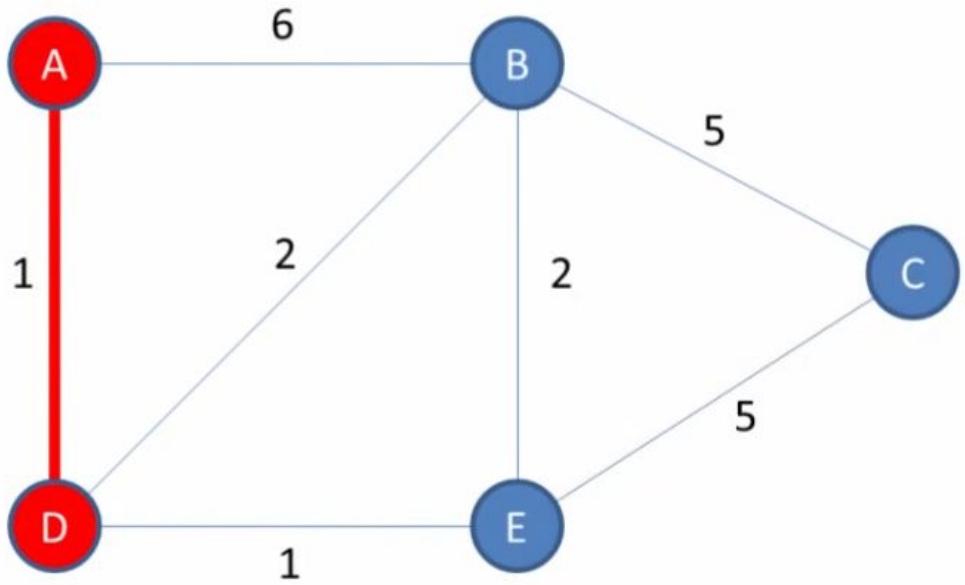
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

A to C



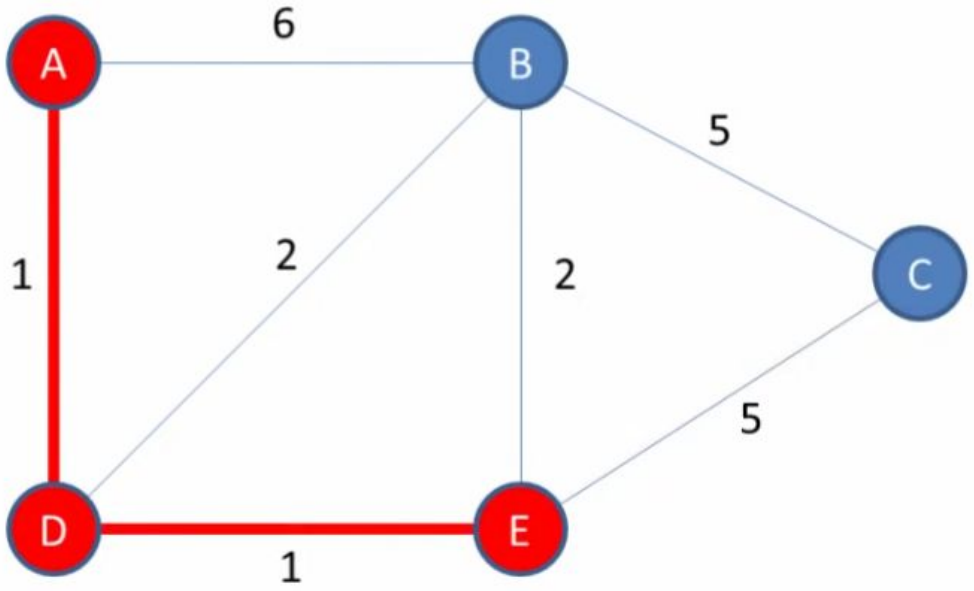
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

A to D



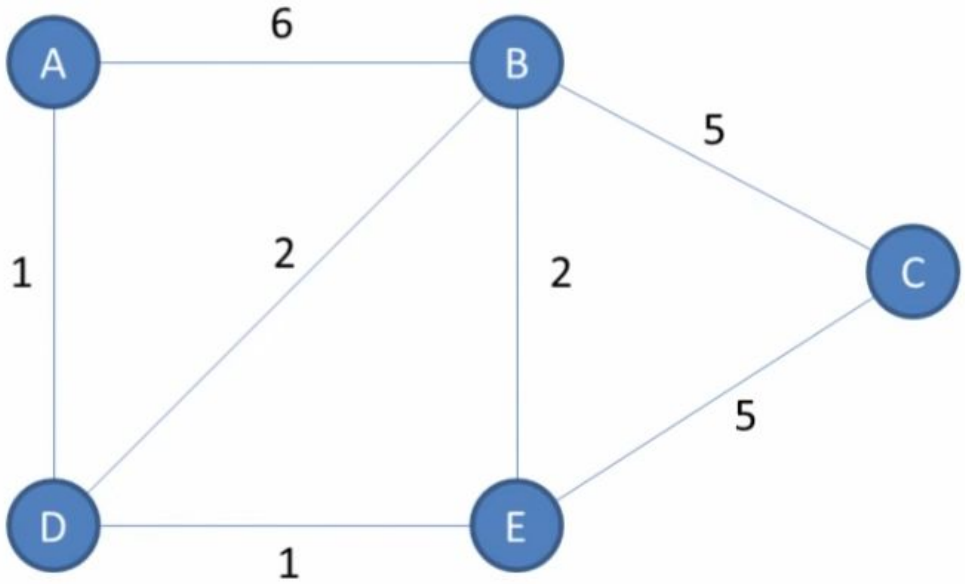
vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

A to E



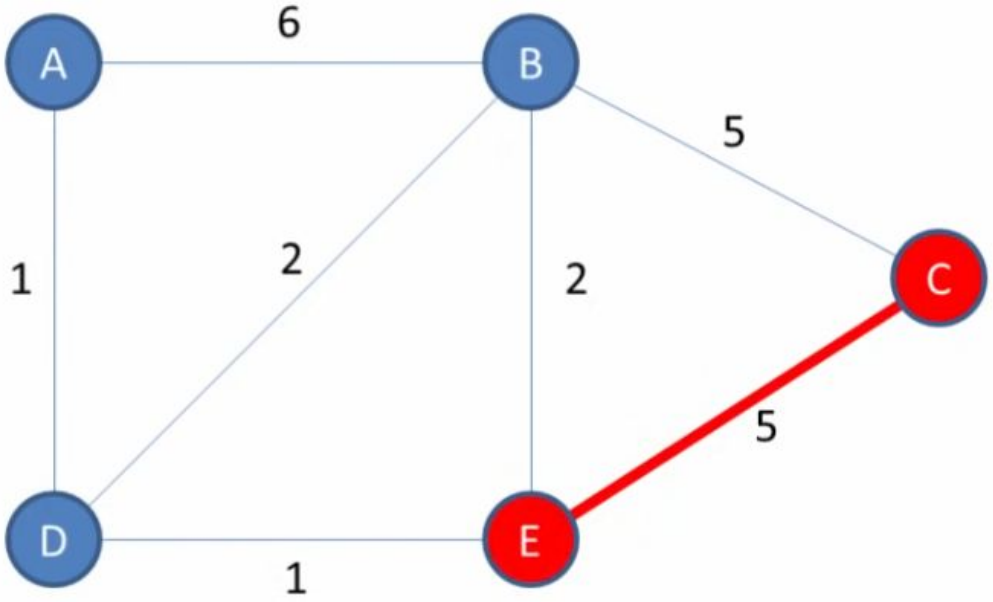
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We also have the shortest sequence of vertices from A to every other vertex



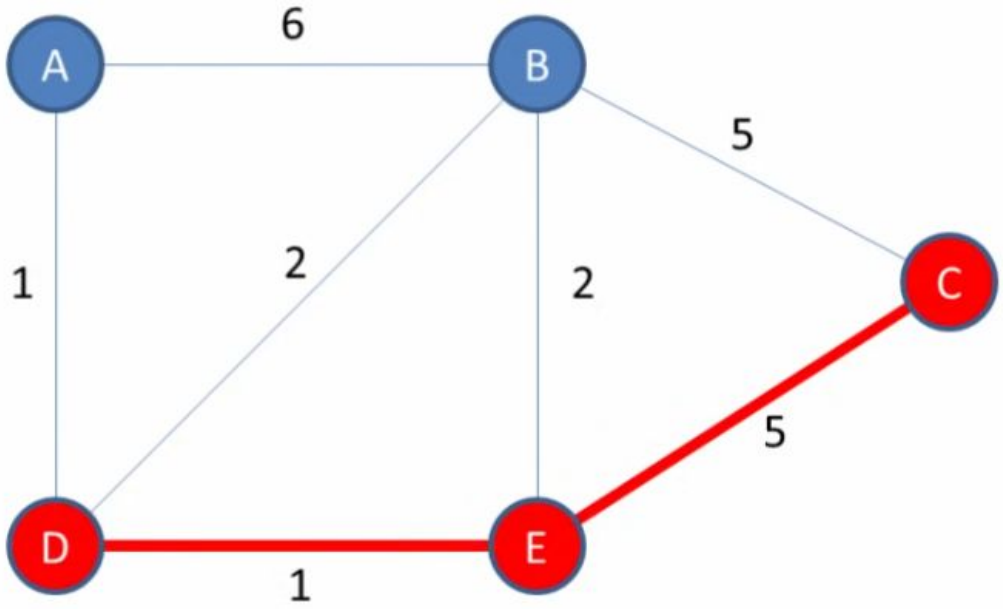
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We arrived at C via E Path = E → C



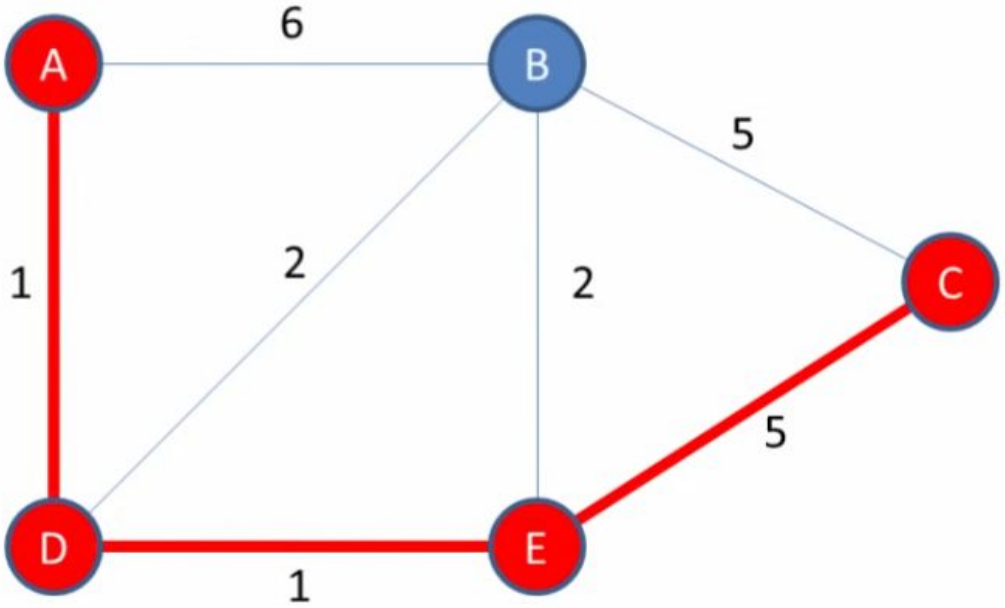
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We arrived at E via D Path = D → E → C



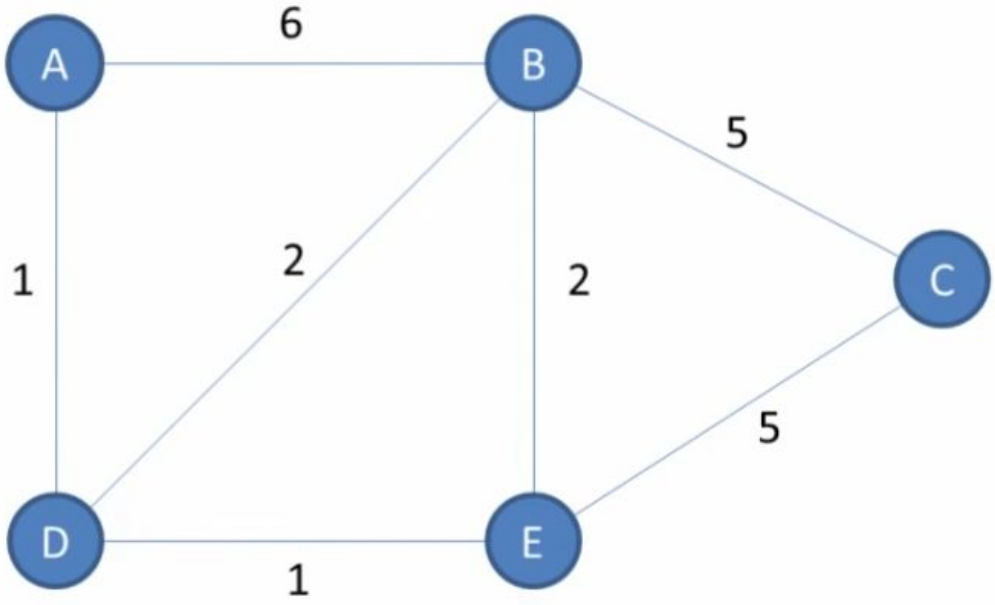
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We arrived at D via A Path = A → D → E → C

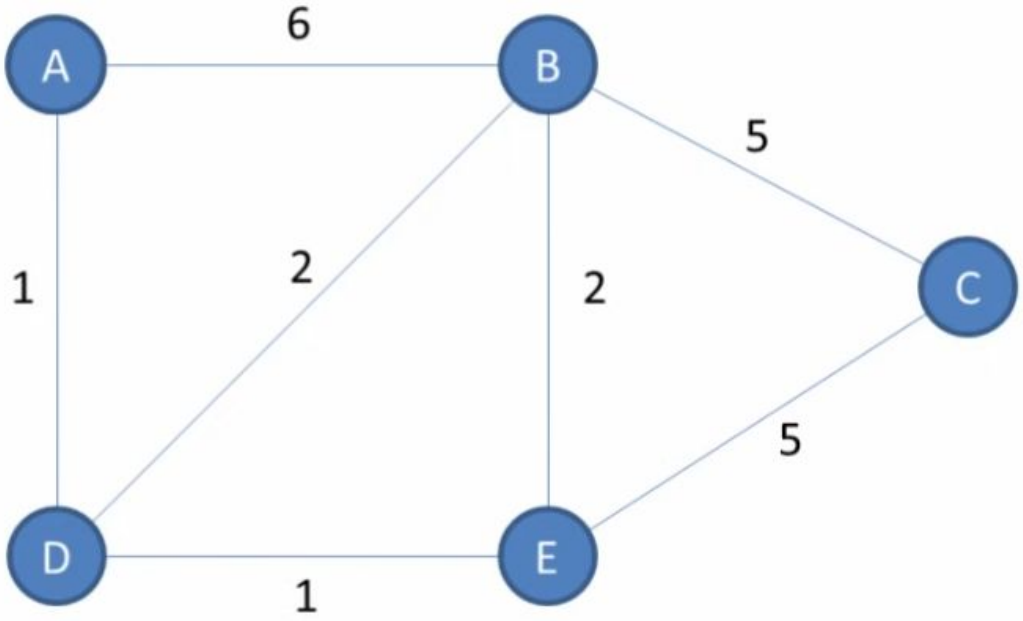


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Dijkstra's Shortest Path Algorithm



Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

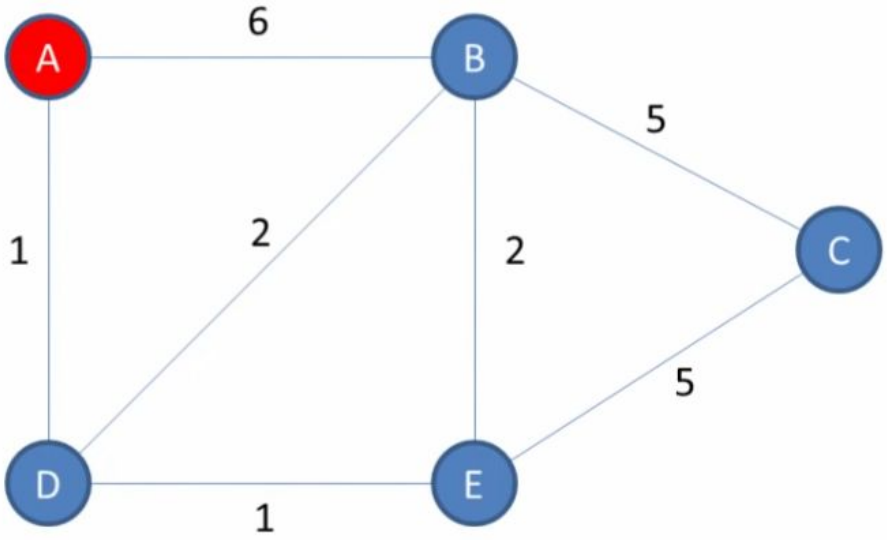


Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

Visited = [ ]

Unvisited = [A, B, C, D, E]

Consider the start vertex, A
Distance to A from A = 0
Distances to all other vertices from A are unknown, therefore ∞ (infinity)

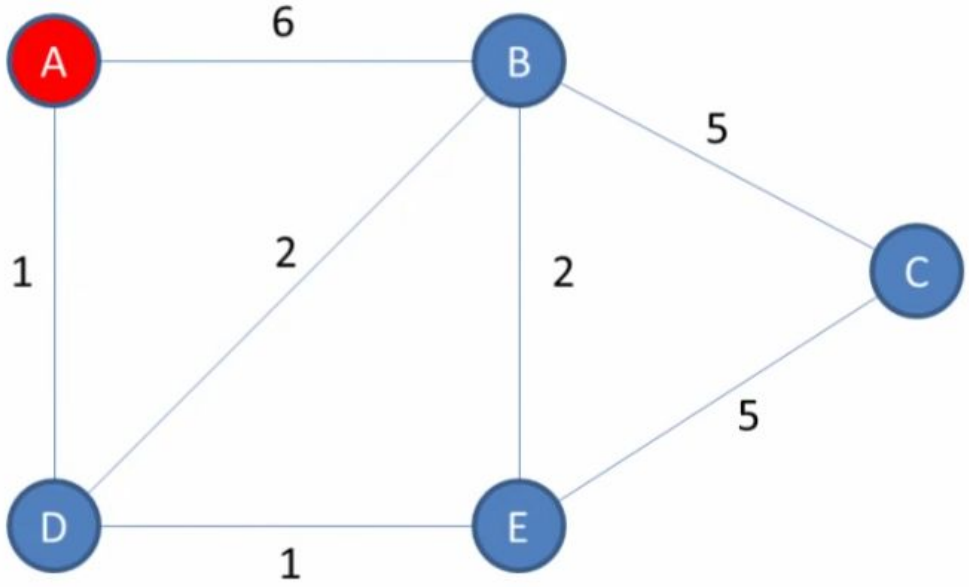


Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

Visited = [ ]

Unvisited = [A, B, C, D, E]

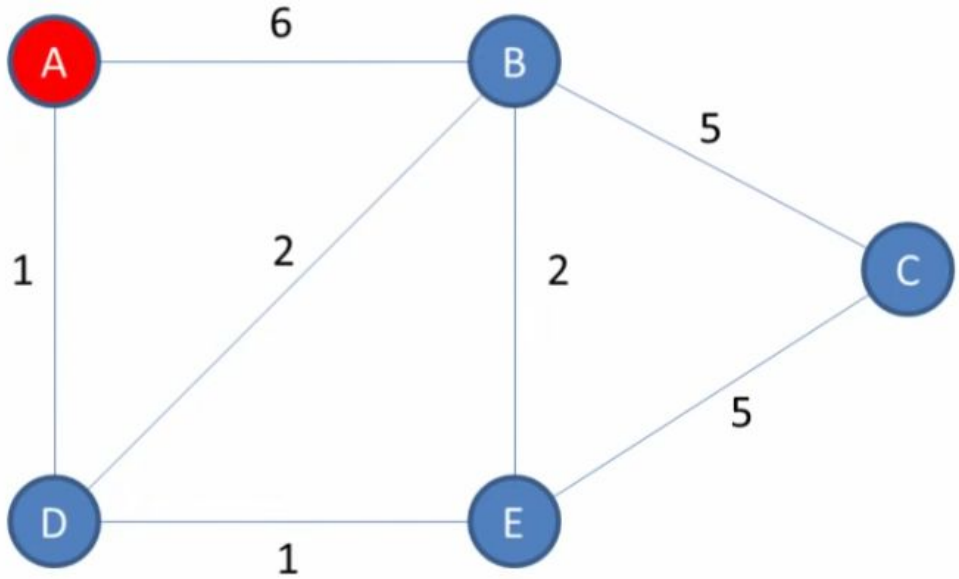
Consider the start vertex, A
Distance to A from A = 0
Distances to all other vertices from A are unknown, therefore ∞ (infinity)



Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visit the unvisited vertex with the smallest known distance from the start vertex

First time around, this is the start vertex itself, A



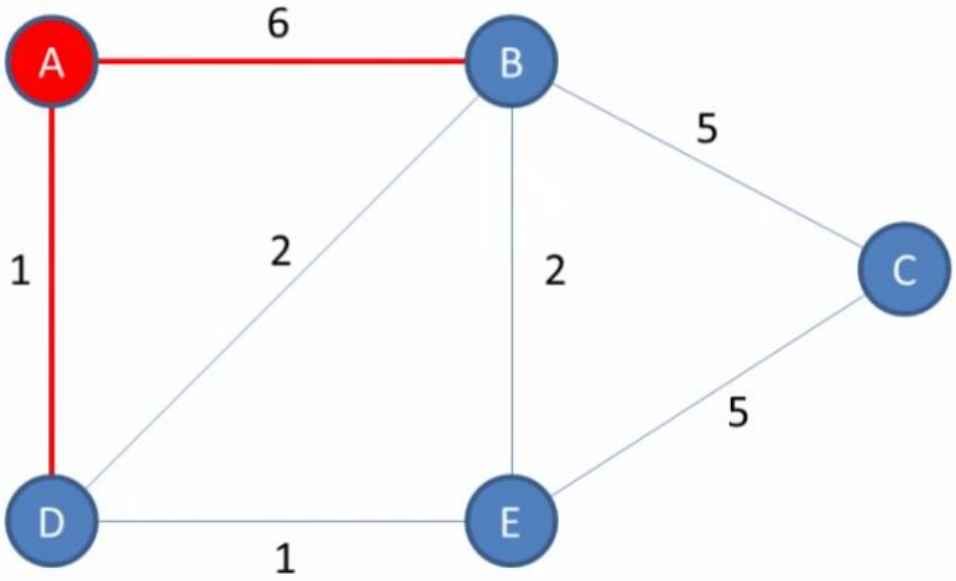
Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E]

For the current vertex, examine its unvisited neighbours

We are currently visiting A and its unvisited neighbours are B and D

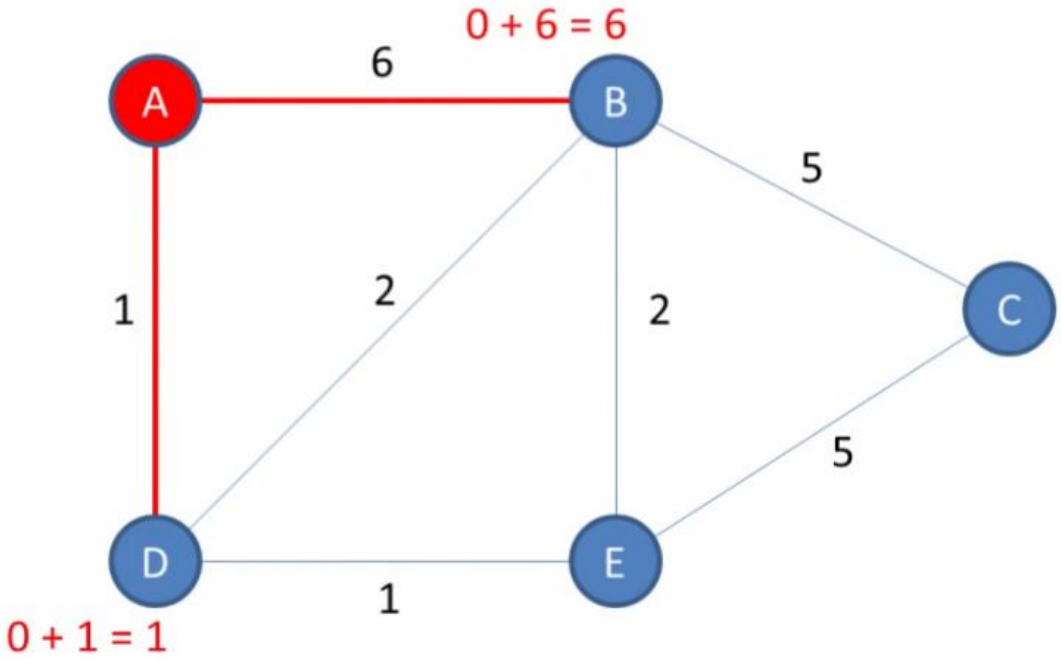


Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E]

For the current vertex, calculate the distance of each neighbour from the start vertex

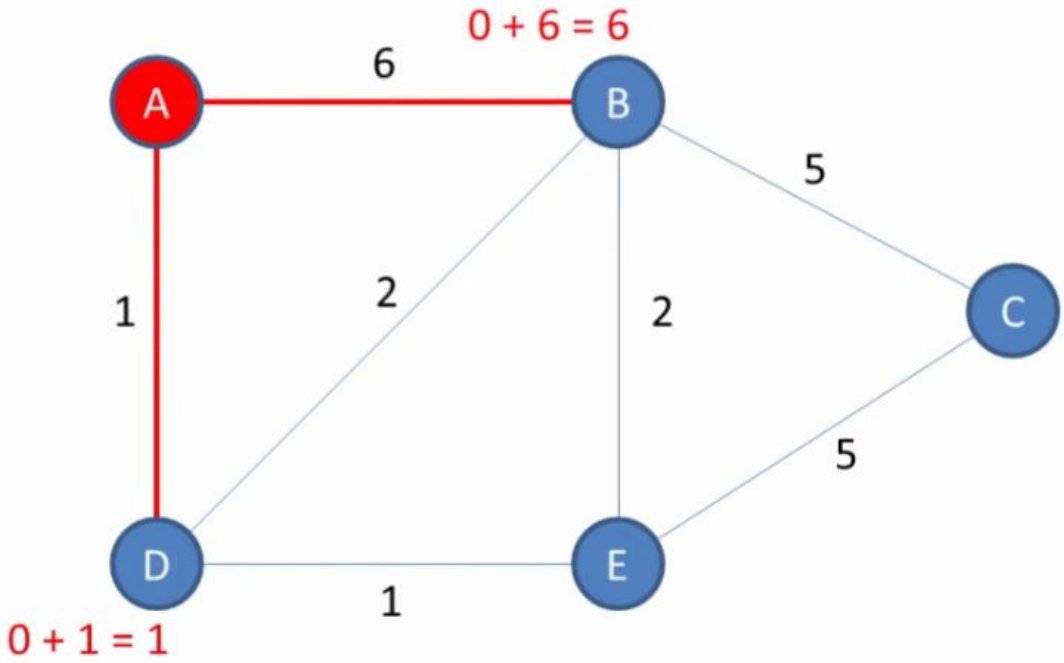


Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E]

If the calculated distance of a vertex is less than the known distance, update the shortest distance



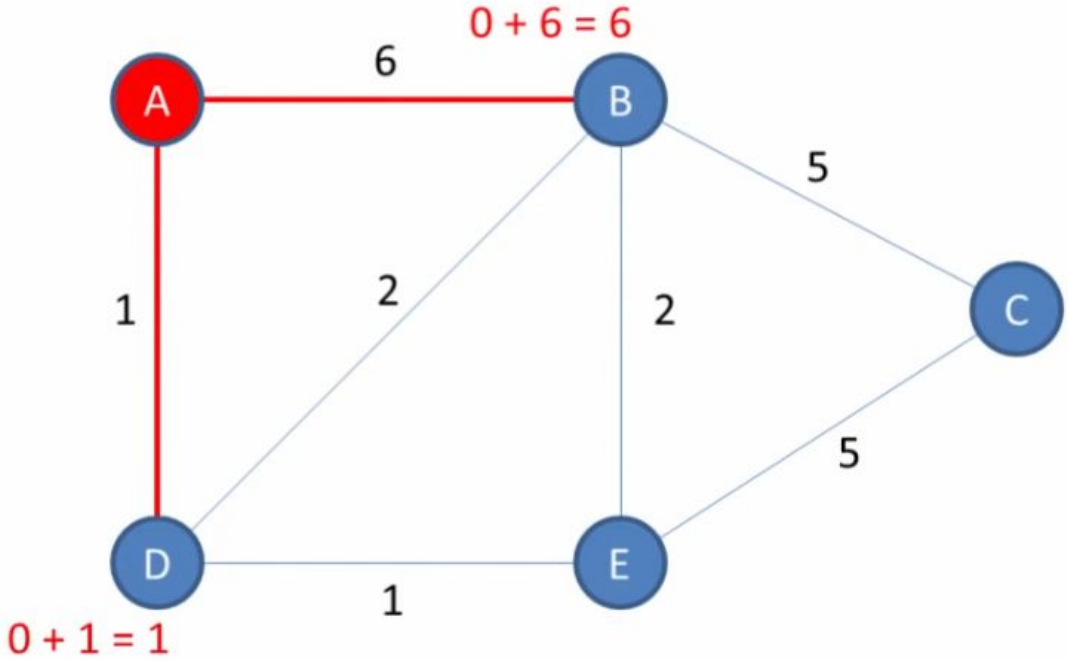
Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E]

Update the previous vertex for each of the updated distances

In this case we visited B and D via A

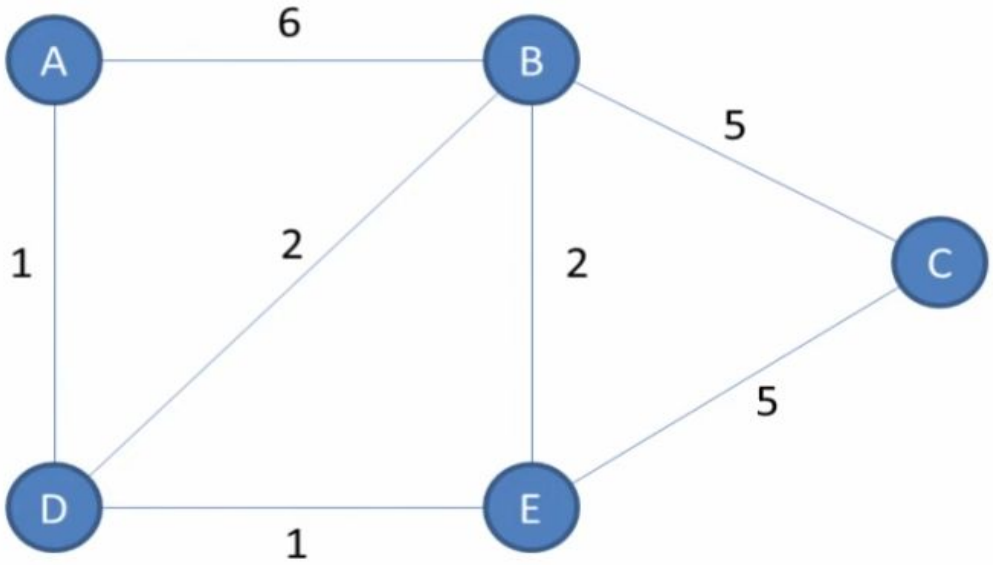


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E]

Add the current vertex to the list of visited vertices



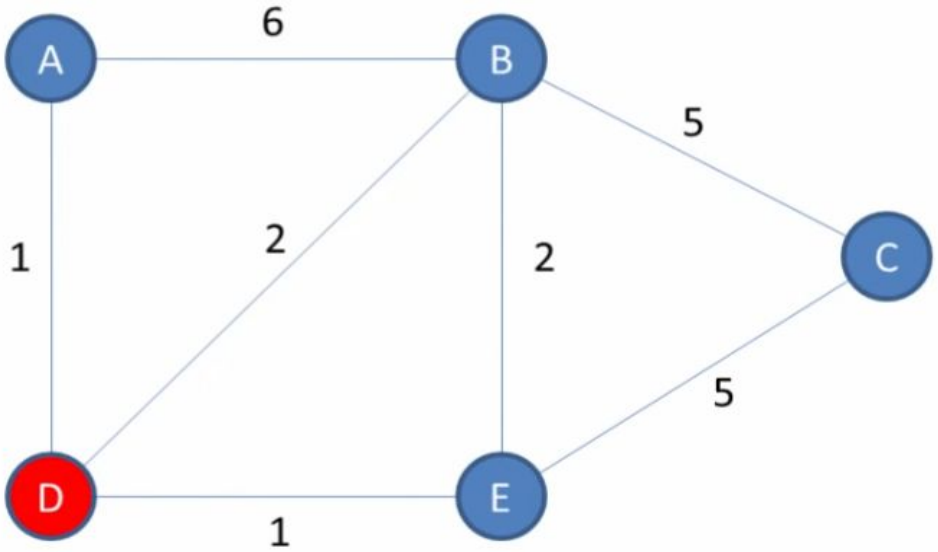
Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

Visit the unvisited vertex with the smallest known distance from the start vertex

This time around, it is vertex D



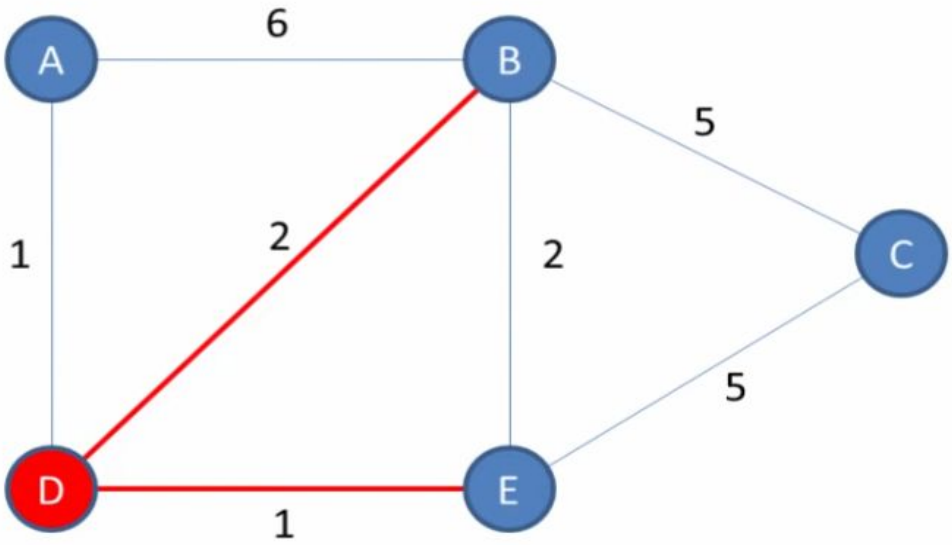
Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

For the current vertex, examine its unvisited neighbours

We are currently visiting D and its unvisited neighbours are B and E

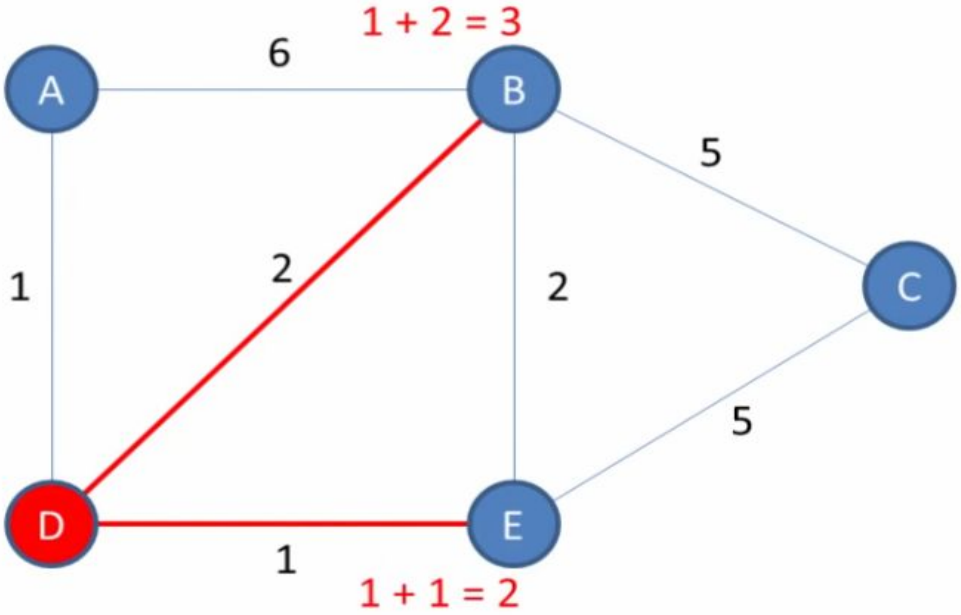


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

For the current vertex, calculate the distance of each neighbour from the start vertex

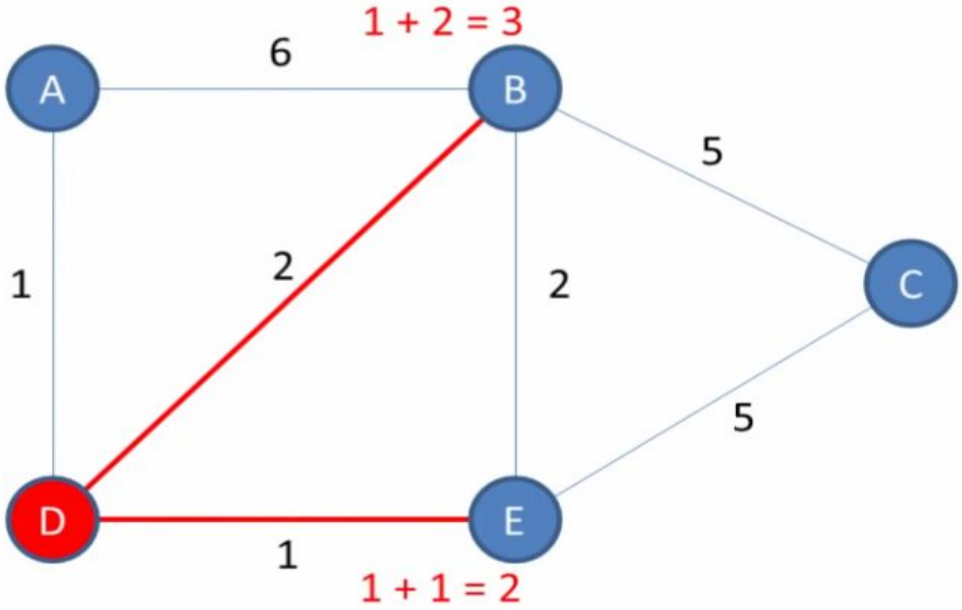


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

If the calculated distance of a vertex is less than the known distance, update the shortest distance



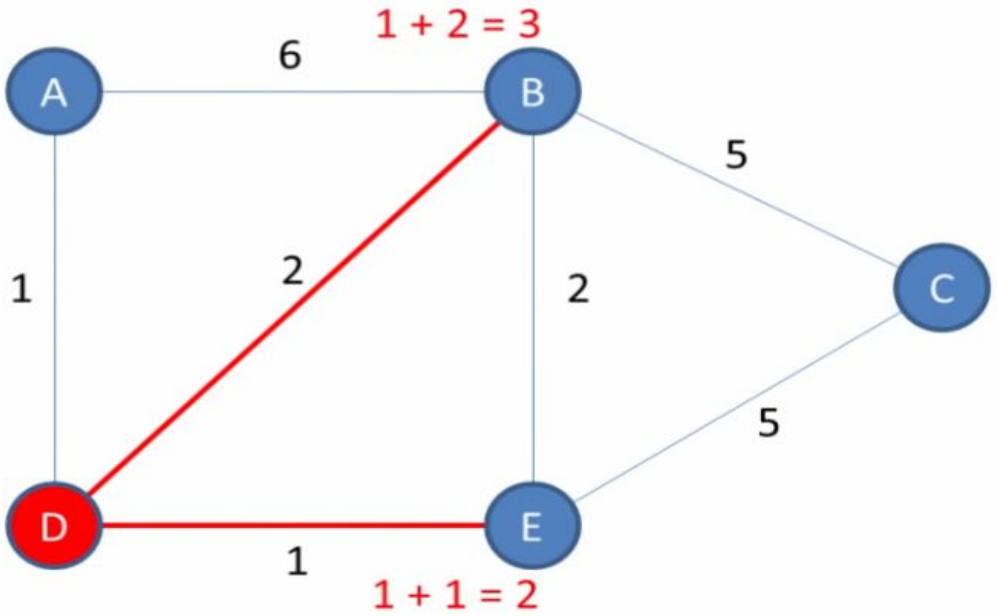
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	A
C	∞	
D	1	A
E	2	

Visited = [A]

Unvisited = [B, C, D, E]

Update the previous vertex for each of the updated distances

In this case we visited B and E via D

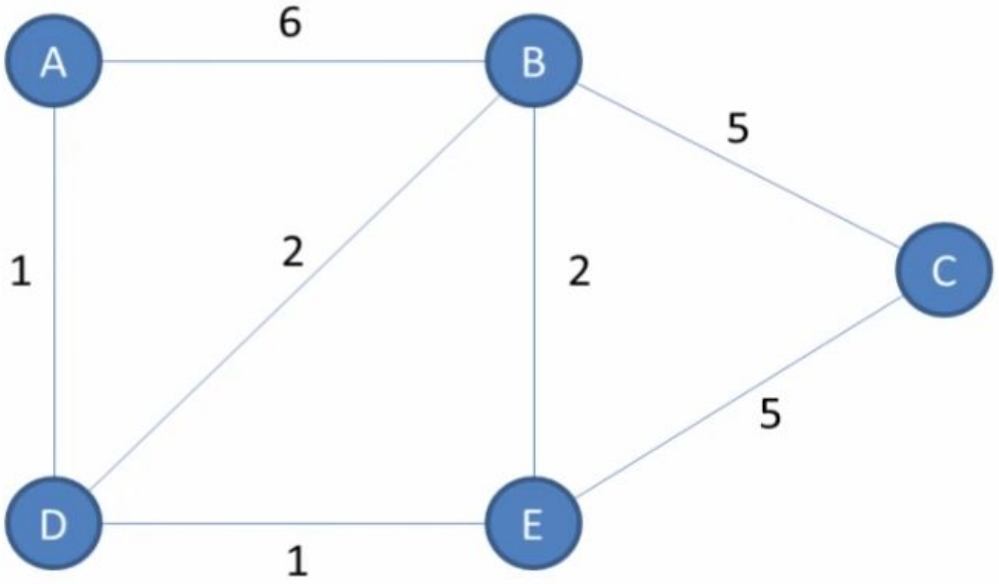


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A]

Unvisited = [B, C, D, E]

Add the current vertex to the list of visited vertices



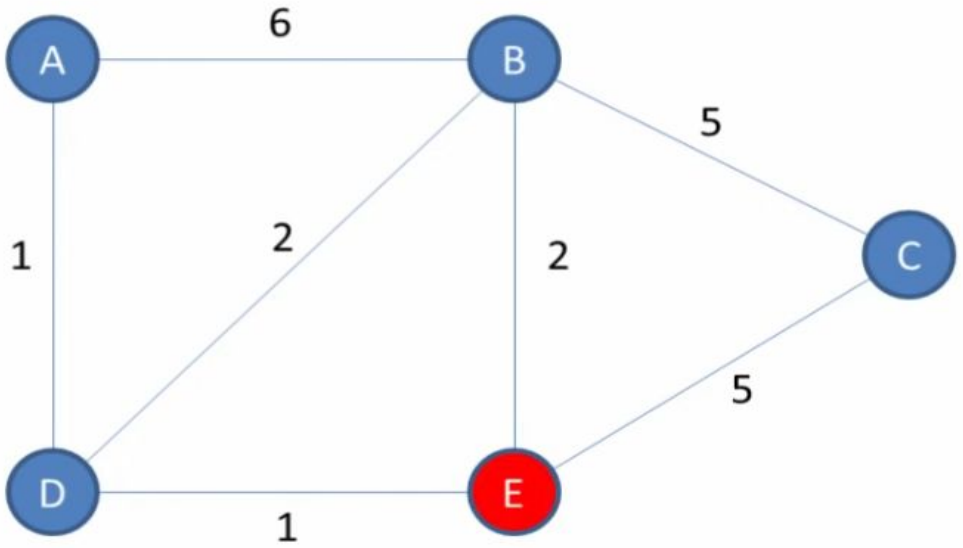
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

Visit the unvisited vertex with the smallest known distance from the start vertex

This time around, it is vertex E



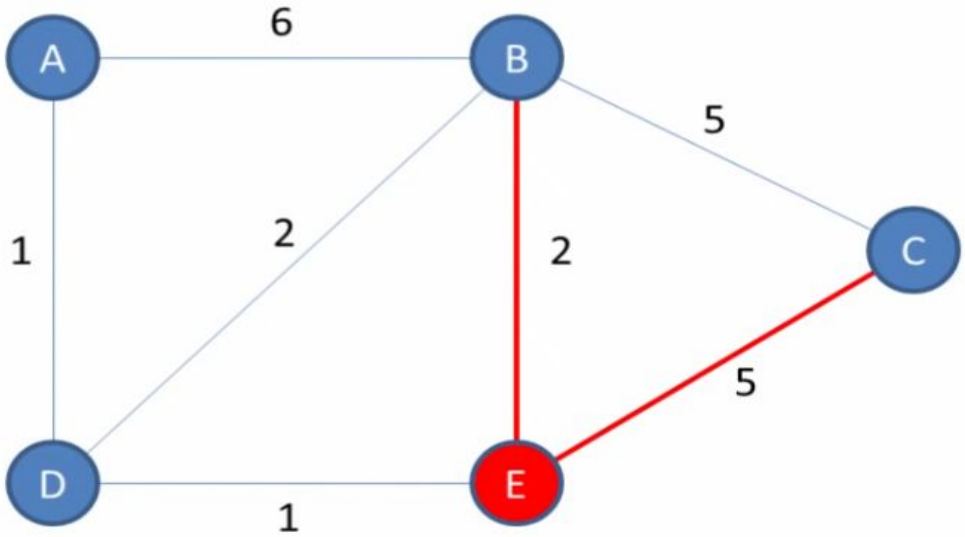
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

For the current vertex, examine its unvisited neighbours

We are currently visiting E and its unvisited neighbours are B and C

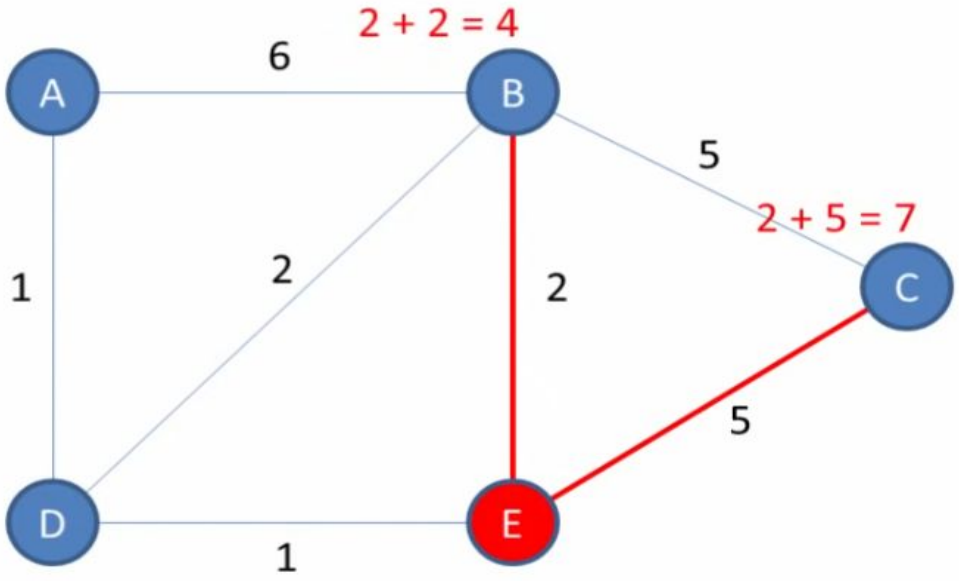


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

For the current vertex, calculate the distance of each neighbour from the start vertex



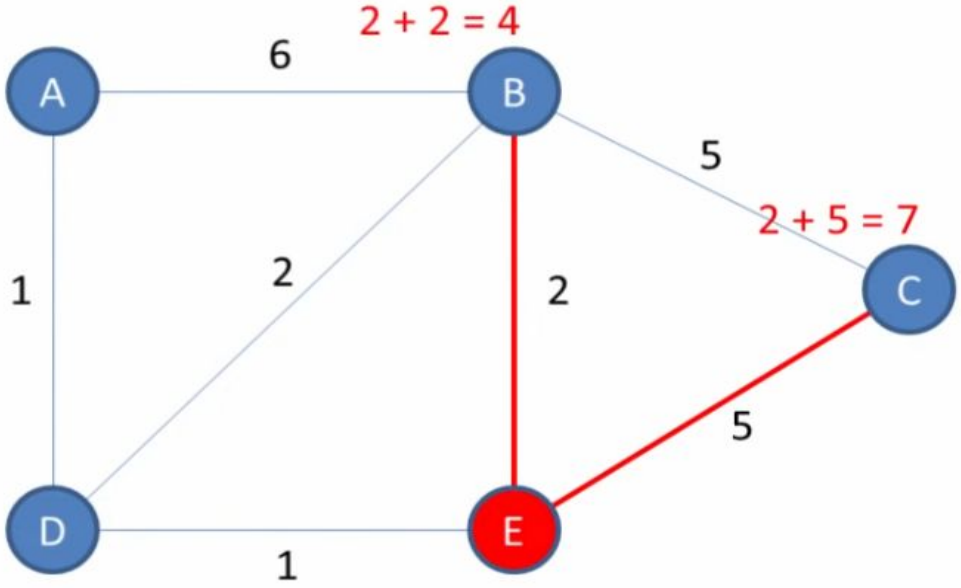
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

If the calculated distance of a vertex is less than the known distance, update the shortest distance

We do not need to update the distance to B



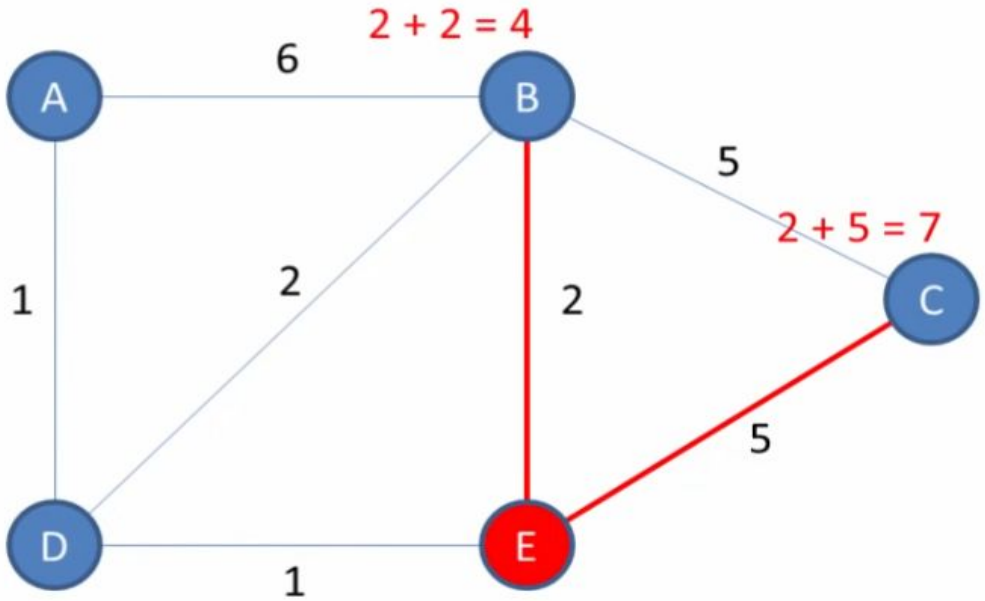
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

Update the previous vertex for each of the updated distances

In this case we visited C via E



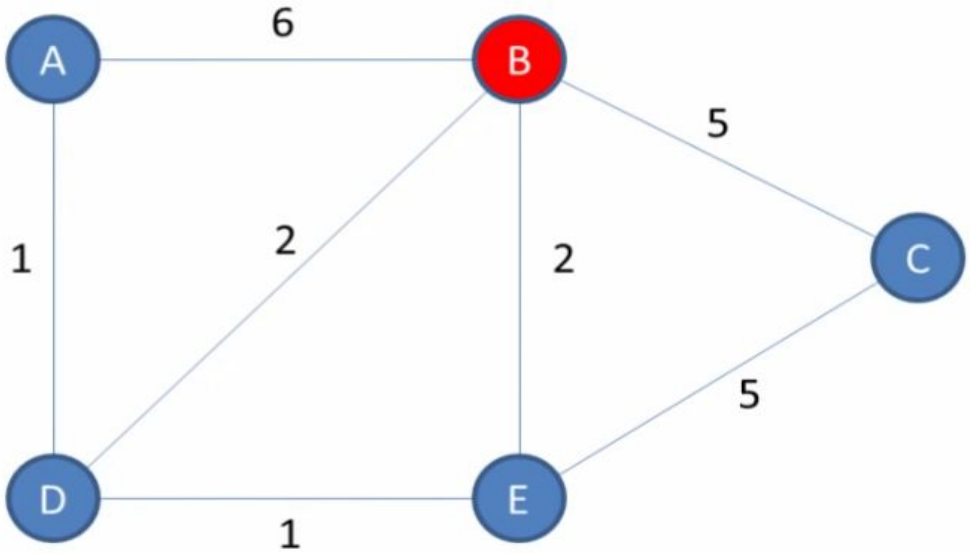
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

Visit the unvisited vertex with the smallest known distance from the start vertex

This time around, it is vertex B



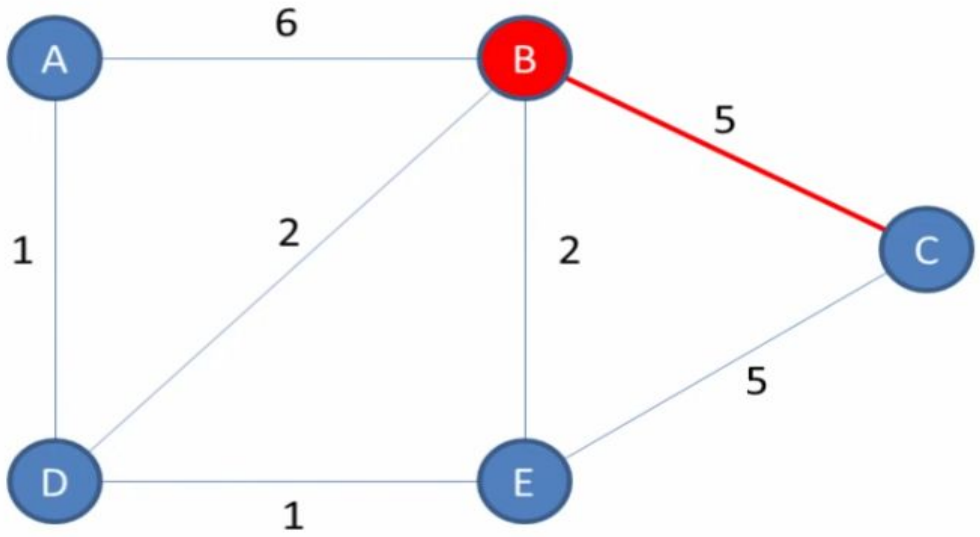
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

For the current vertex, examine its unvisited neighbours

We are currently visiting B and its only unvisited neighbour is C

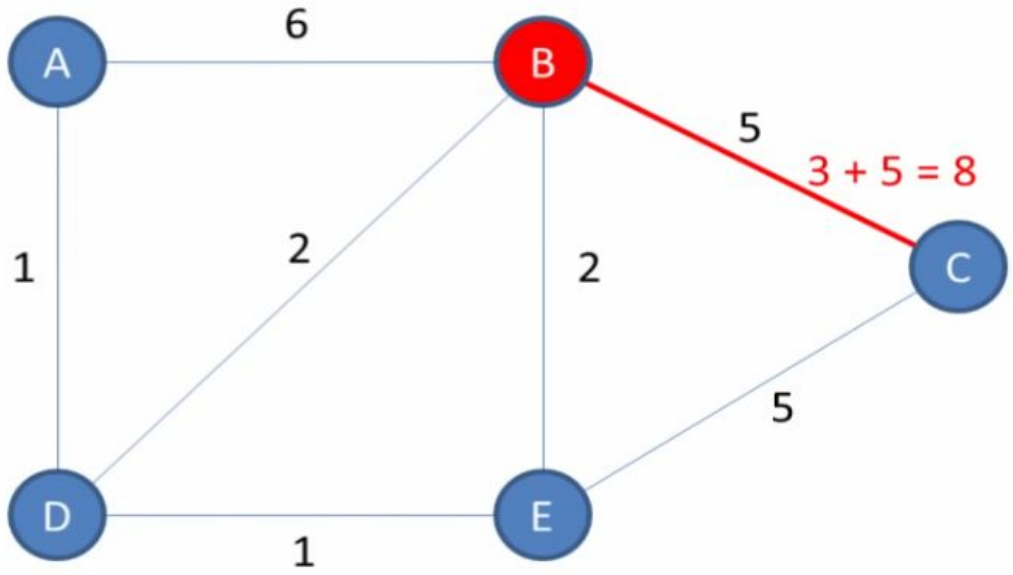


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

For the current vertex, calculate the distance of each neighbour from the start vertex



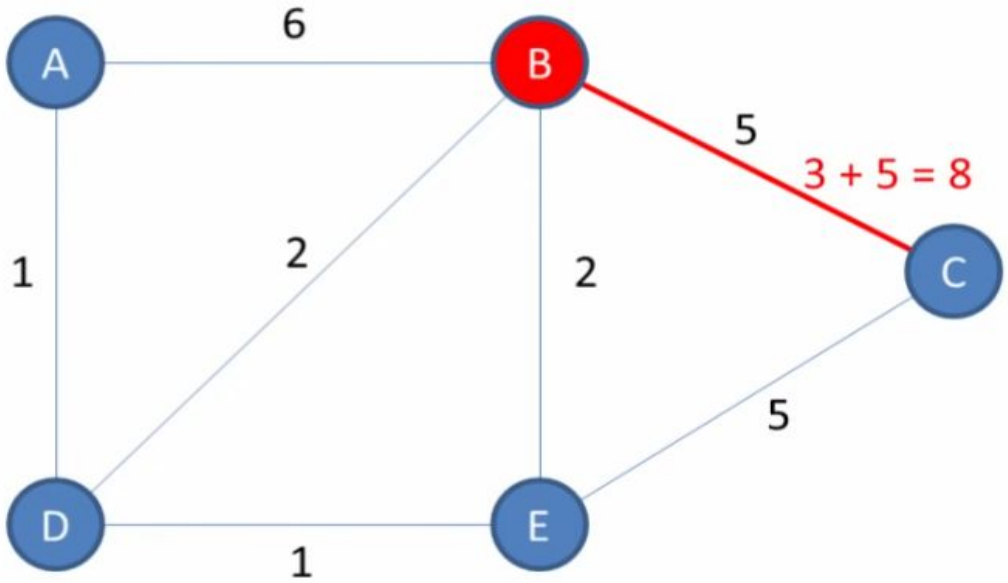
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

If the calculated distance of a vertex is less than the known distance, update the shortest distance

We do not need to update the distance to C

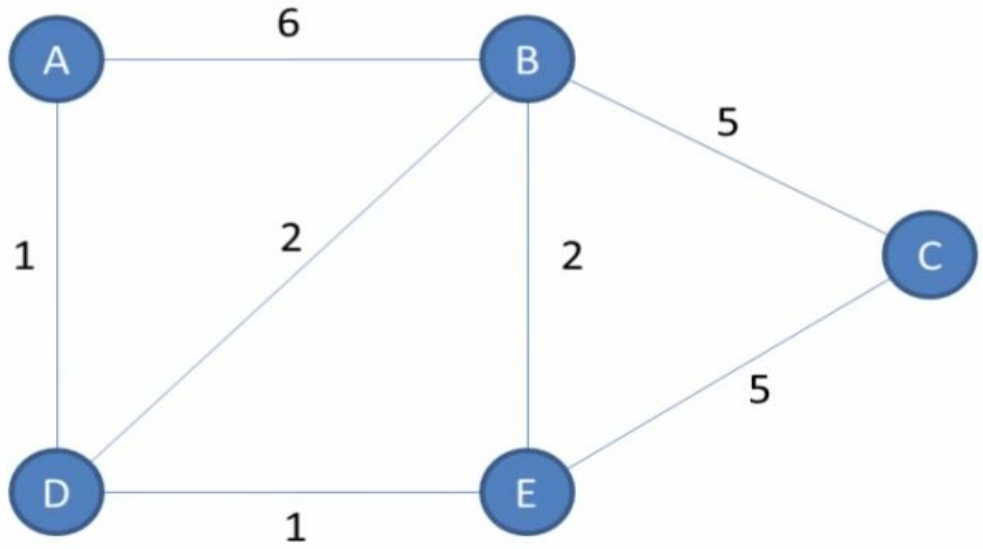


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

Add the current vertex to the list of visited vertices

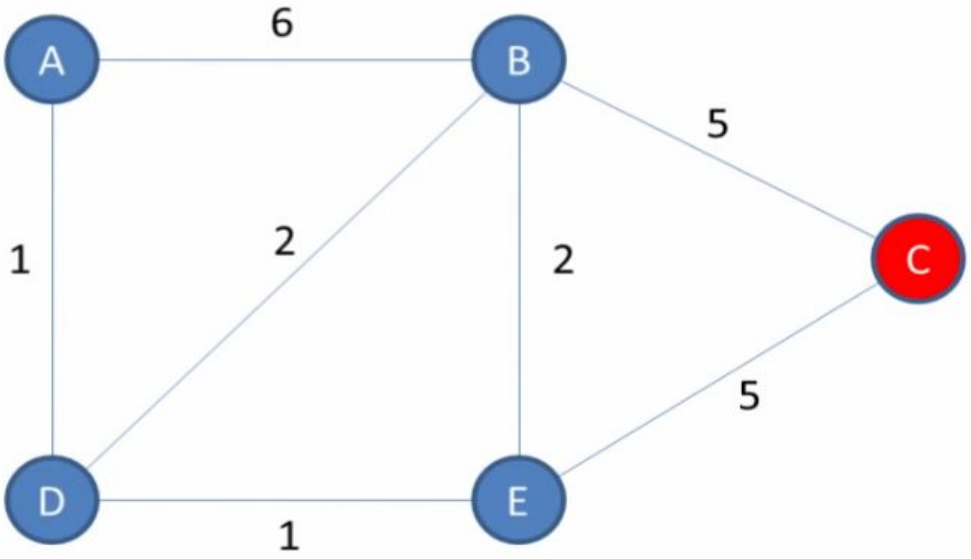


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B] Unvisited = [C]

For the current vertex, examine its unvisited neighbours

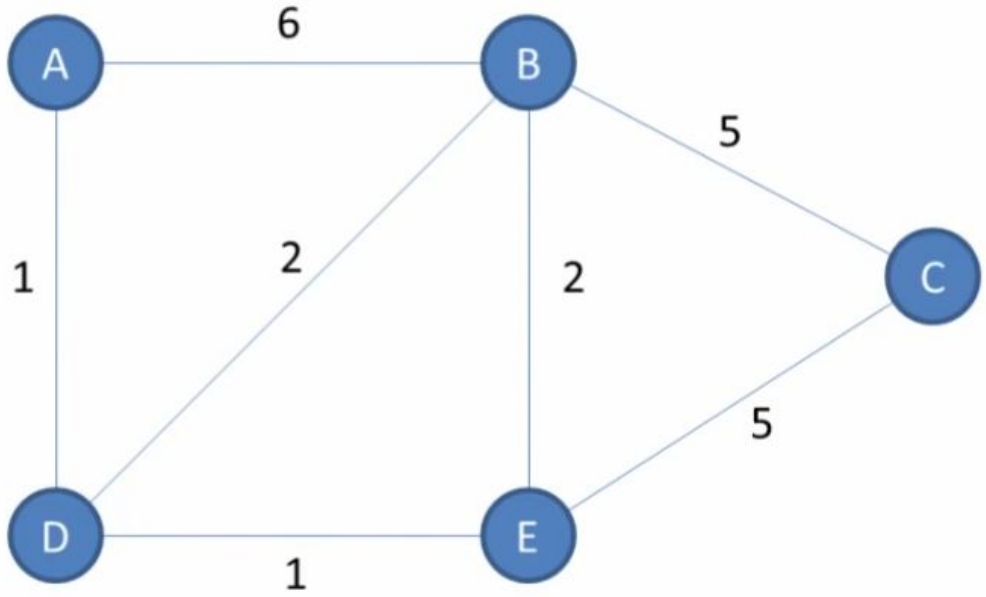
We are currently visiting C and it has no unvisited neighbours



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B] Unvisited = [C]

Add the current vertex to the list of visited vertices



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B, C] Unvisited = []

Algorithm

Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start = ∞ (infinity)

Repeat
- Visit the unvisited vertex with the smallest known distance from the start vertex
- For the current vertex, examine its unvisited neighbours
- For the current vertex, calculate distance of each neighbour from start vertex
- If the calculated distance of a vertex is less than the known distance, update the shortest distance
- Update the previous vertex for each of the updated distances
- Add the current vertex to the list of visited vertices

Until all vertices visited

Flow-based routing

Flow-based routing is a **static routing algorithm** in computer networks that uses **load and topology** to determine a route.

It requires some information to be known in advance, such as the subnet topology

In flow routing, flow refers to a ***single, meaningful, end-to-end activity*** over the network.

Another way to define flow is as ***a stream of IP packets*** that move from a particular source IP address and port to a destination IP address and port.

Flow-based routing

Flow-based routing is a network computing technique that ***groups packets into flows*** and routes them as a single unit.

It's useful for protecting certain types of traffic from others.

How it works

- Flow-based routing groups packets into flows *based on characteristics like source and destination IP addresses, ports, or protocol*.
- It uses topology and load to decide on a route.
- It uses alternative congestion control schemes to protect certain types of traffic.

When it's useful

- Flow-based routing is useful when certain types of traffic need to be protected from others. For example, it can protect web access from peer-to-peer communications.

Flooding

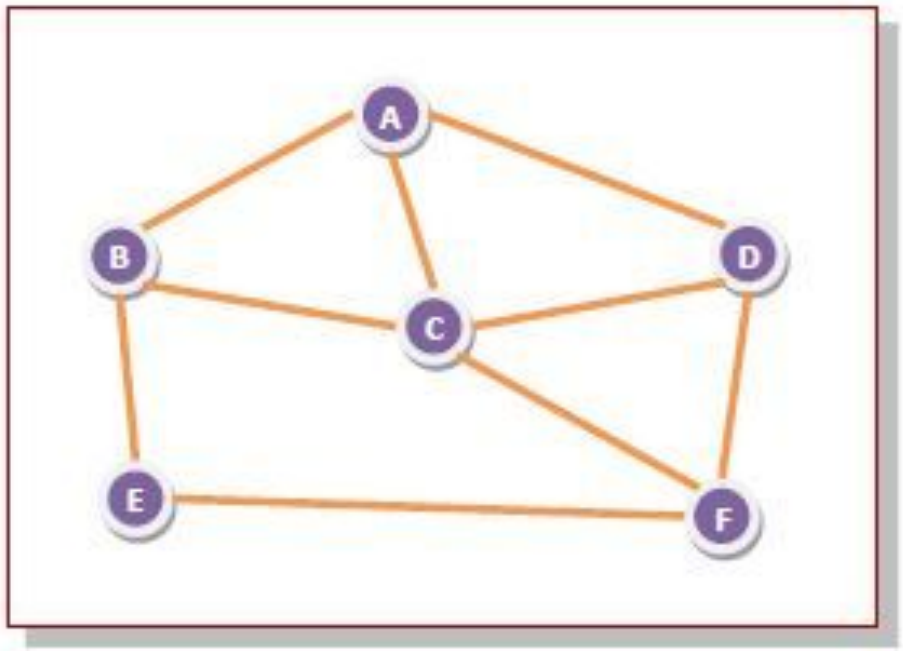
Why Flooding

- When a routing algorithm is implemented,
  - Each router must make decisions based on local knowledge,
  - not the complete picture of the network.

What is Flooding

- **Flooding is a *non-adaptive routing* technique**
  - **when a data packet arrives at a router,**
  - **it is sent to all the outgoing links except the one it has arrived on.**

Example..... "flooding technique –



- An incoming packet to A, will be sent to B, C and D.
- B will send the packet to C and E.
- C will send the packet to A, D and F.
- D will send the packet to C and F, receiving from A
- D will send the packet to A and F, receiving from C
- E will send the packet to F.
- F will send the packet to C and E (if from D).
- F will send the packet to D and E (if from C).

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 that is the shortest.
- The shortest path is always chosen by flooding.
- It is very simple to setup and implement, since a router may know only its neighbours.
- It is extremely robust.

- Even in case of malfunctioning of a large number of routers, the packets find a way to reach the destination.
- All nodes that are directly or indirectly connected are visited.
- So, there are no chances for any node to be left out.
- This is a main criteria in the case of broadcast messages.

Limitations –

- Flooding generates a vast number of duplicate packets
- A suitable damping mechanism must be used
- It is wasteful if a single destination needs the packet since it delivers the data packet to all nodes, irrespective of the destination.
- The network may be clogged with unwanted and duplicate data packets.
- This may hamper delivery of other data packets.

Hierarchical Routing

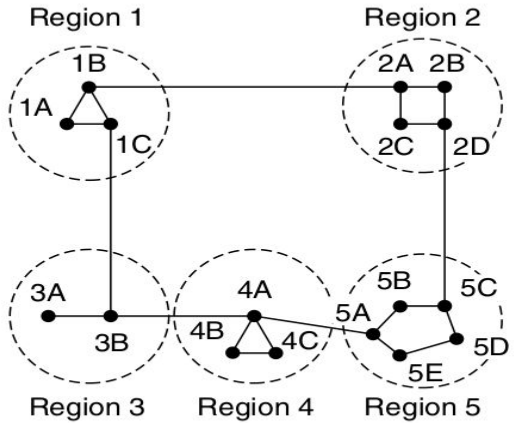
Hierarchical Routing

- As networks grow in size, the router routing tables grow proportionally.
- Not only is router *memory consumed* by ever-increasing tables, but *more CPU time* is needed to scan them and *more bandwidth* is needed to send status reports about them.
- At a certain point, the network may grow to the point where it is no longer feasible for every router to have an entry for every other router.
    - so the routing will have to be done *hierarchically*

- **When hierarchical routing is used, the routers are divided into regions.**
- **Each router *knows all the details* about how to route packets to destinations within its own region but knows nothing about the internal structure of other regions.**

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

Example.....



(a)

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

(b)

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

(c)

Disadvantage:

- **May have increased path length.**
- **For example, the best route from 1A to 5C is via region 2, but with hierarchical routing, all traffic to region 5 goes via region 3, because that is better for most destinations in region 5.**

- **“How many levels should the hierarchy have?”**
  - **For example, consider a network with 720 routers.**
  - **If there is no hierarchy, each router needs 720 routing table entries.**

- **If the network is partitioned into (two-level hierarchy)**
  - **24 regions of 30 routers each,**
  - **each router needs 30 local entries plus 23 remote entries for a total of 53 entries.**

- **If a three-level hierarchy is chosen,**
  - **with 8 clusters each containing 9 regions of 10 routers,**
  - **each router needs 10 entries for local routers,**
  - **8 entries for routing to other regions within its own cluster,**
  - **and 7 entries for distant clusters,**
    - **for a total of 25 entries**