

E- Lecture

Data Structures and Algorithms

By

H R Choudhary (Asstt. Professor)

Department of CSE

Engineering College Ajmer

Topics to be covered

Graph

What do you mean by Graph ?

Graph Representation.

Graph Operation.

Traversing of Graph

- BFS
- DFS

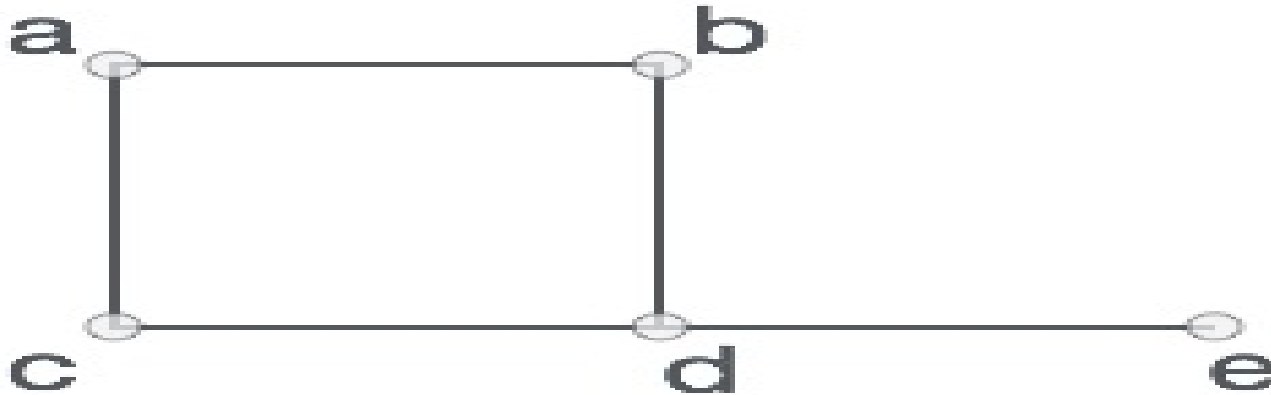
Spanning Tree

Graph

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges.

Formally, a graph is a pair of sets (V, E) , where V is the set of vertices and E is the set of edges, connecting the pairs of vertices. Take a look at the following graph –

Graph..



In the above graph,

$$V = \{a, b, c, d, e\}$$

$$E = \{ab, ac, bd, cd, de\}$$

Graph Representation

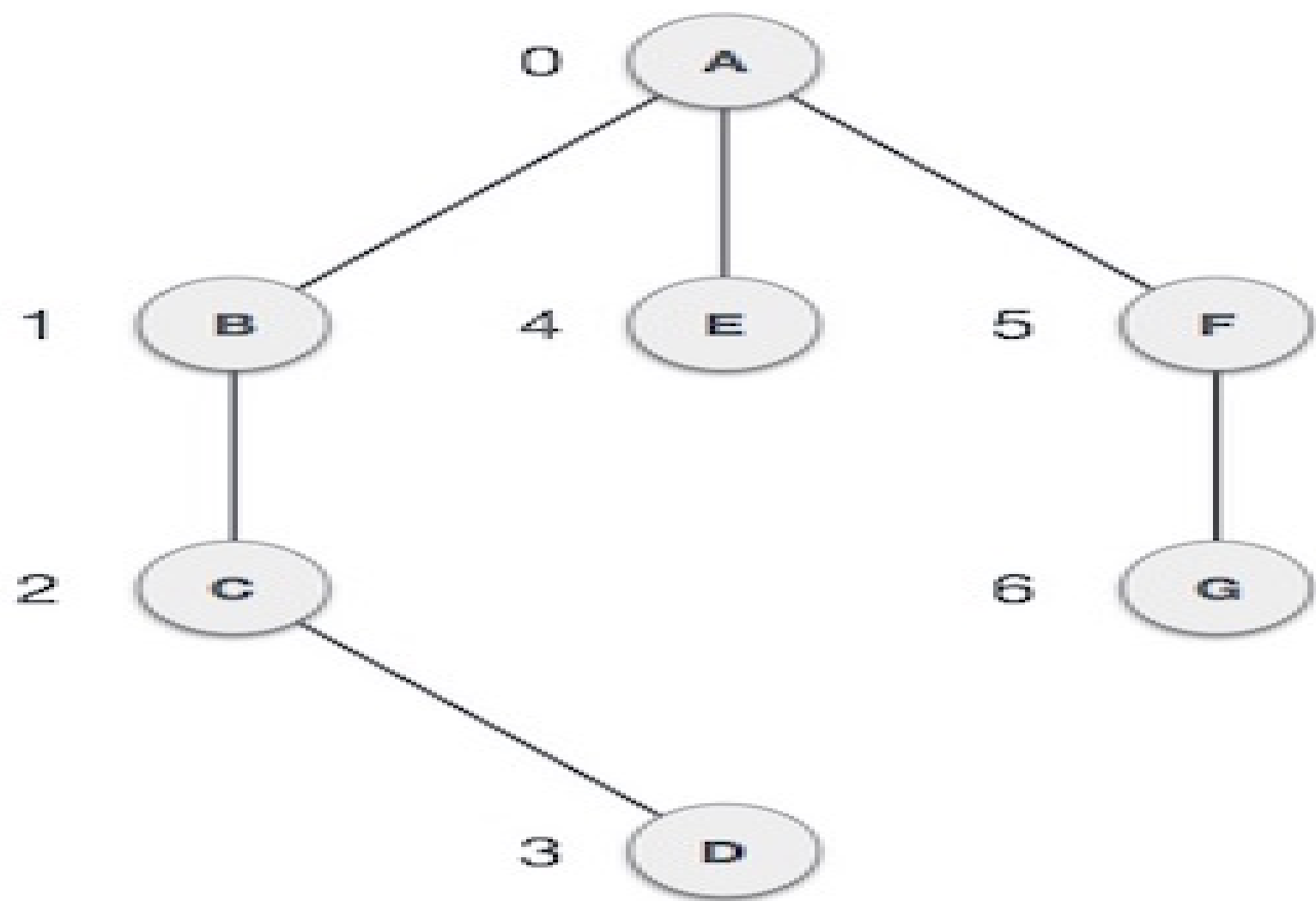
Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms –

Vertex – Each node of the graph is represented as a vertex. In the following example, the labeled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on.

Edge – Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represents edges. We can use a two-dimensional array to represent an array as shown in the following image. Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0.

Adjacency – Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.

Path – Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.



Graph Operation

Basic Operations:

Following are basic primary operations of a Graph –

- Add Vertex – Adds a vertex to the graph.
- Add Edge – Adds an edge between the two vertices of the graph.
- Display Vertex – Displays a vertex of the graph.

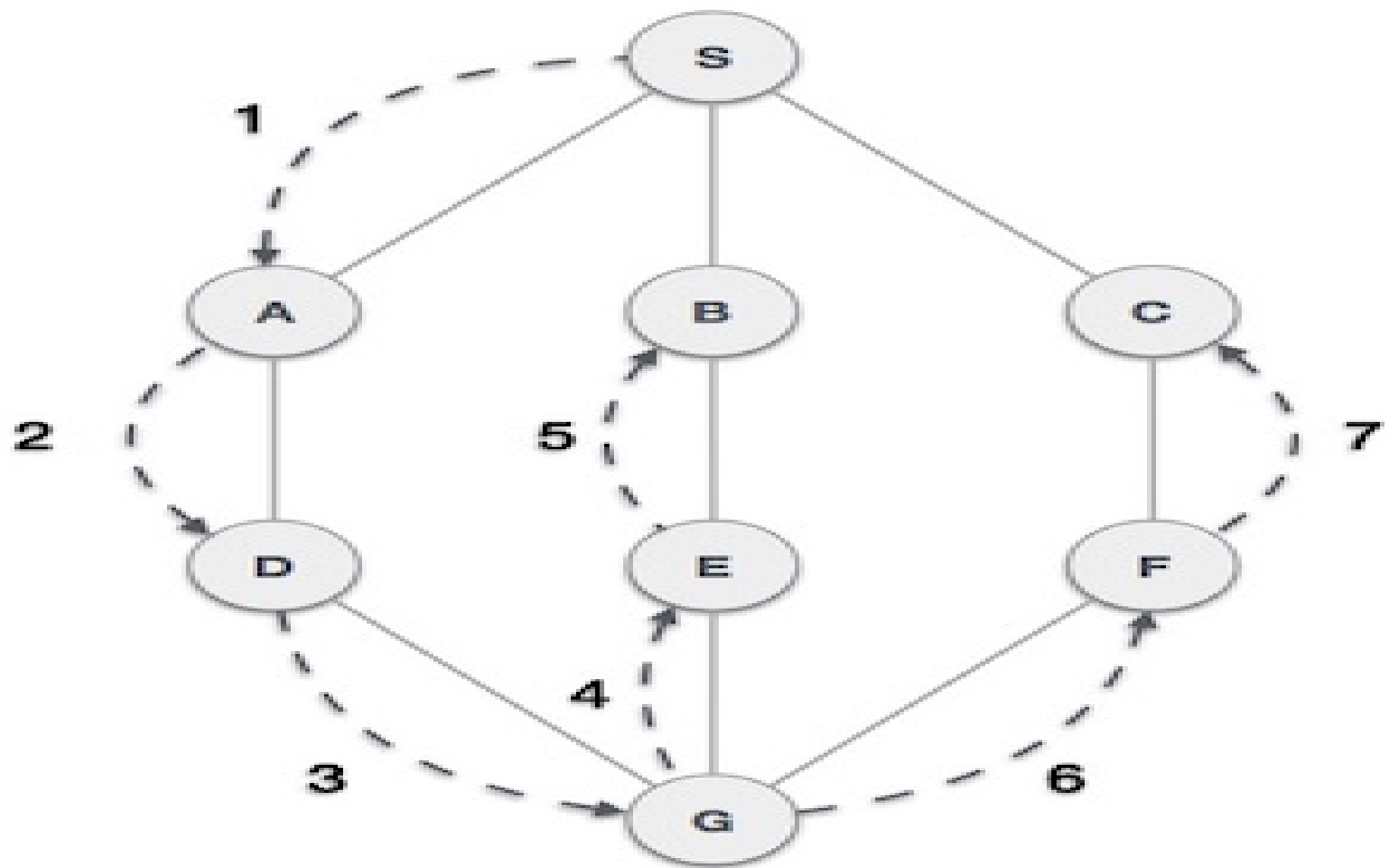
Traversing of Graph

There are two methods to traverse a Graph:

- Depth First Search
- Breadth First Search

Depth First Search

Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As in the example given above, DFS algorithm traverses from S to A to D to G to E to B first, then to F and lastly to C. It employs the following rules.

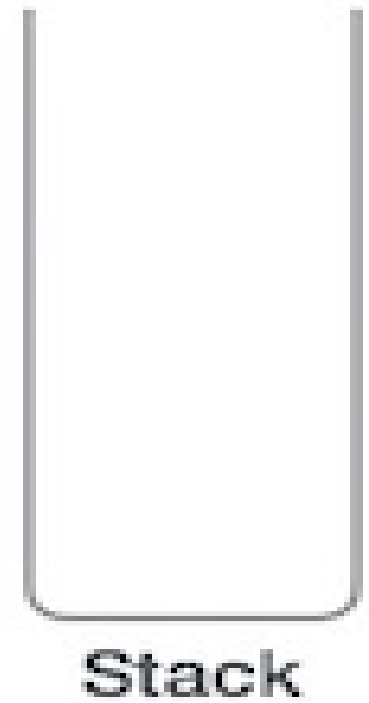
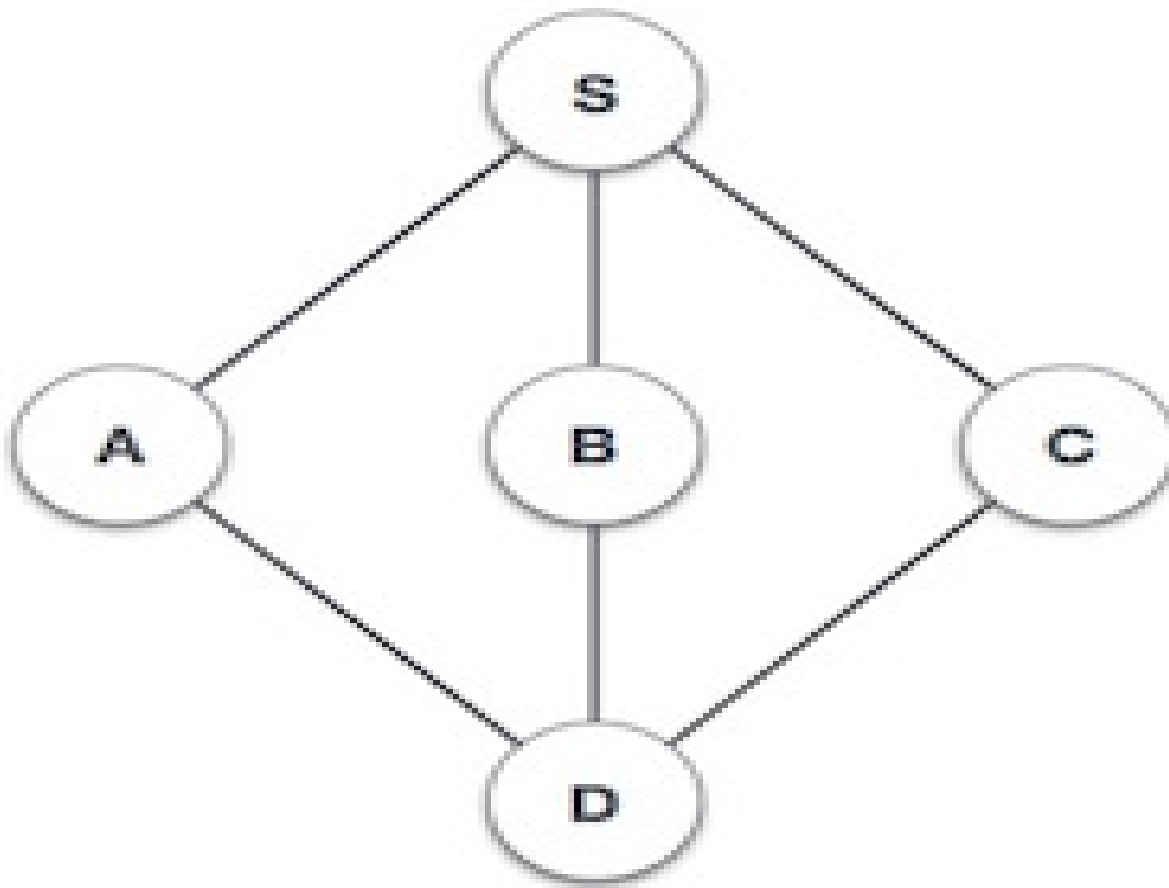
Rule 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

Rule 2 – If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)

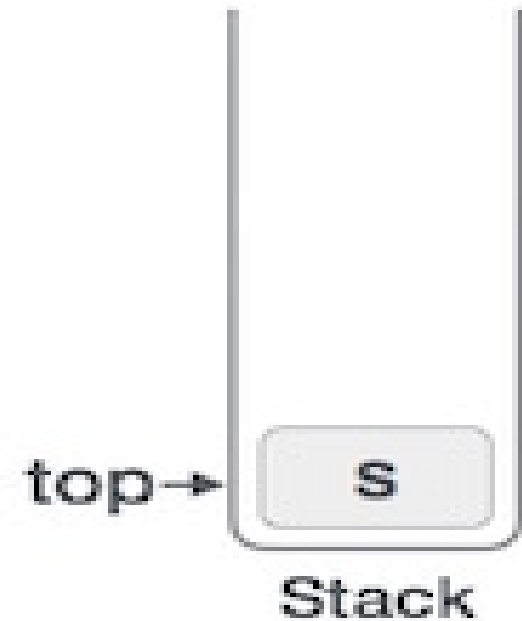
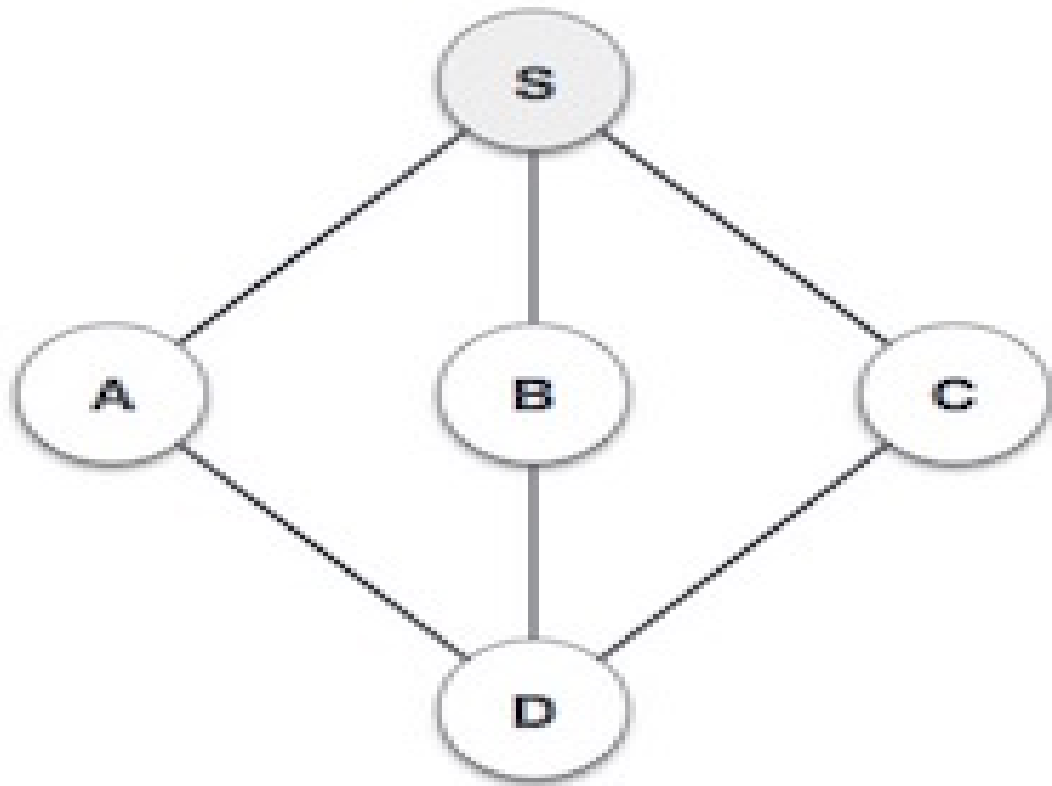
Rule 3 – Repeat Rule 1 and Rule 2 until the stack is empty.

Example..

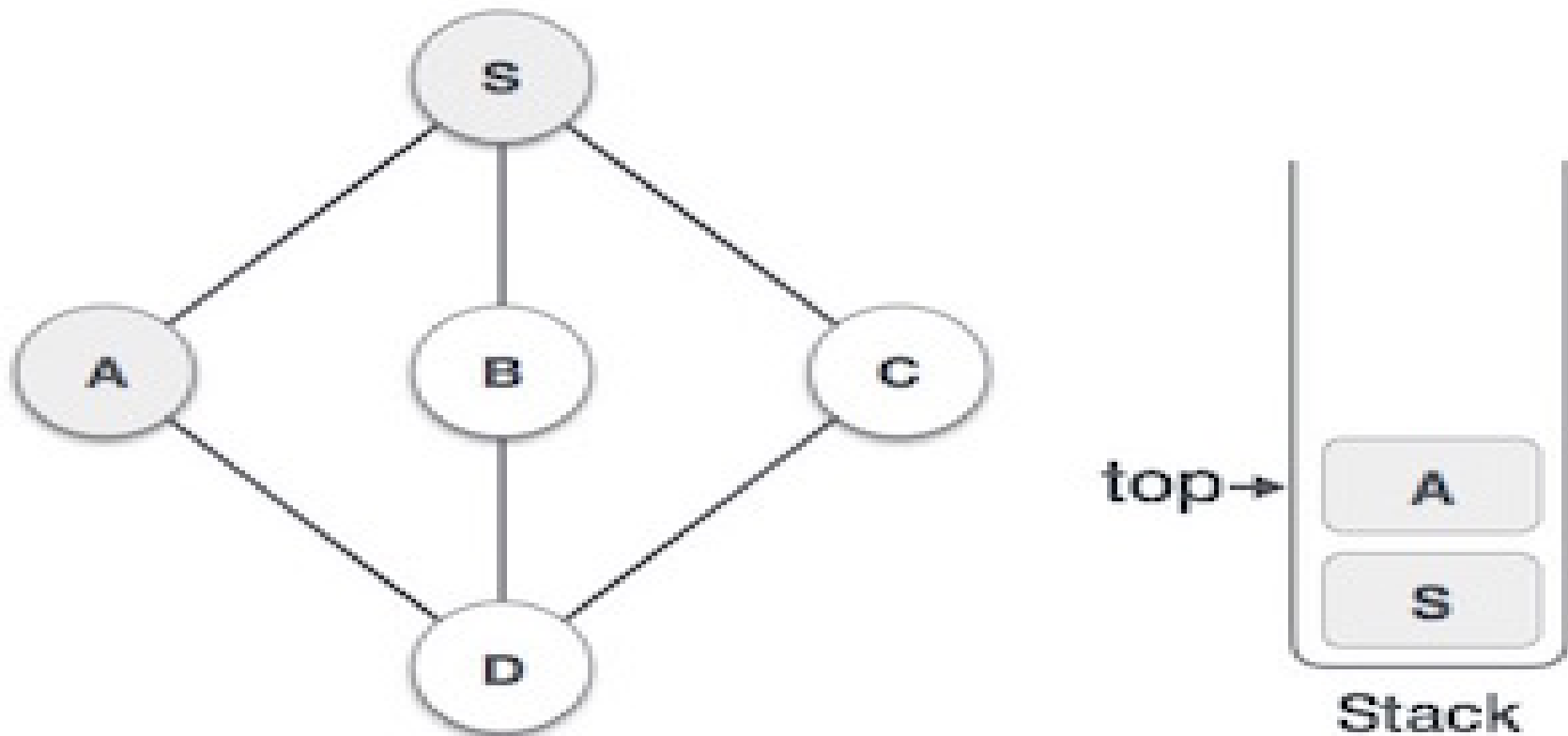
Initialize the stack.



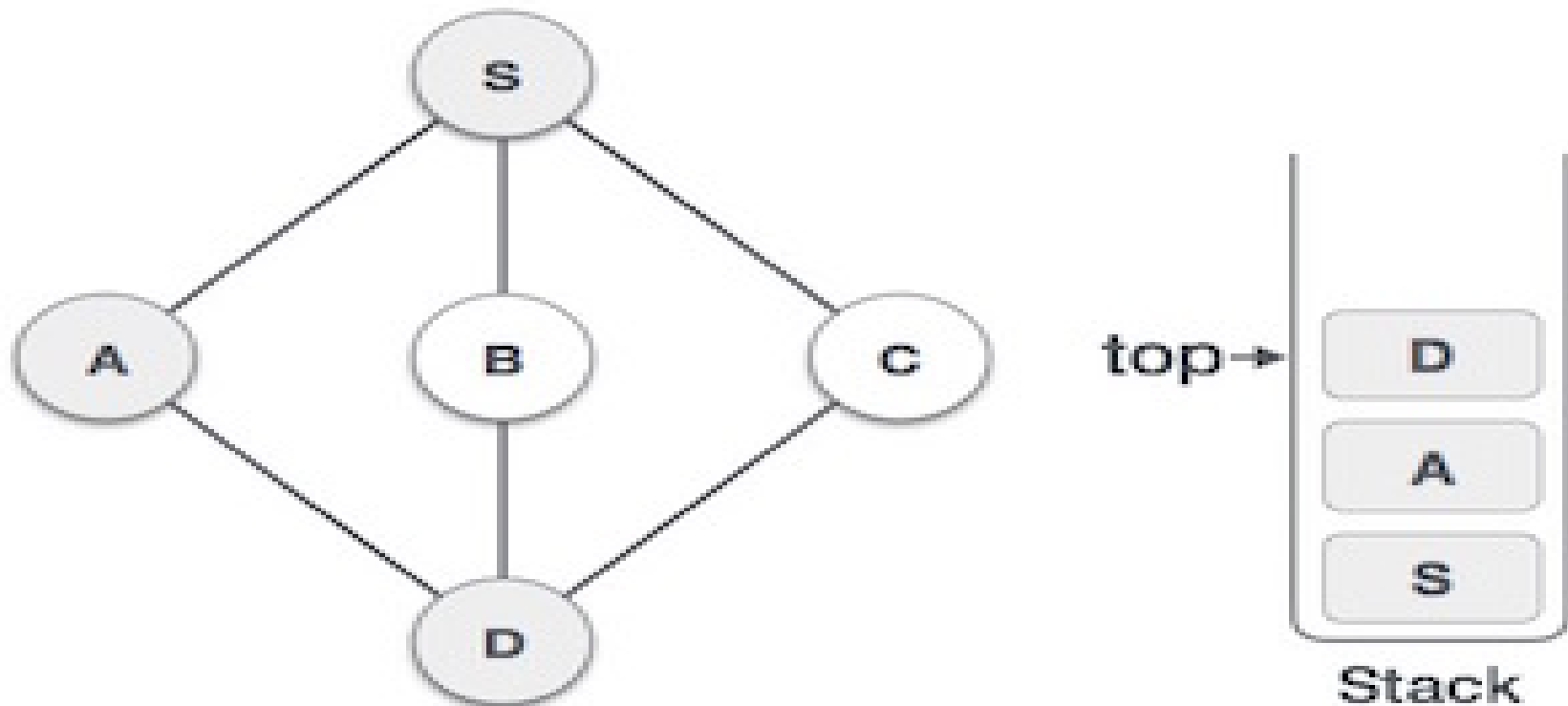
Mark S as visited and put it onto the stack. Explore any unvisited adjacent node from S. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.



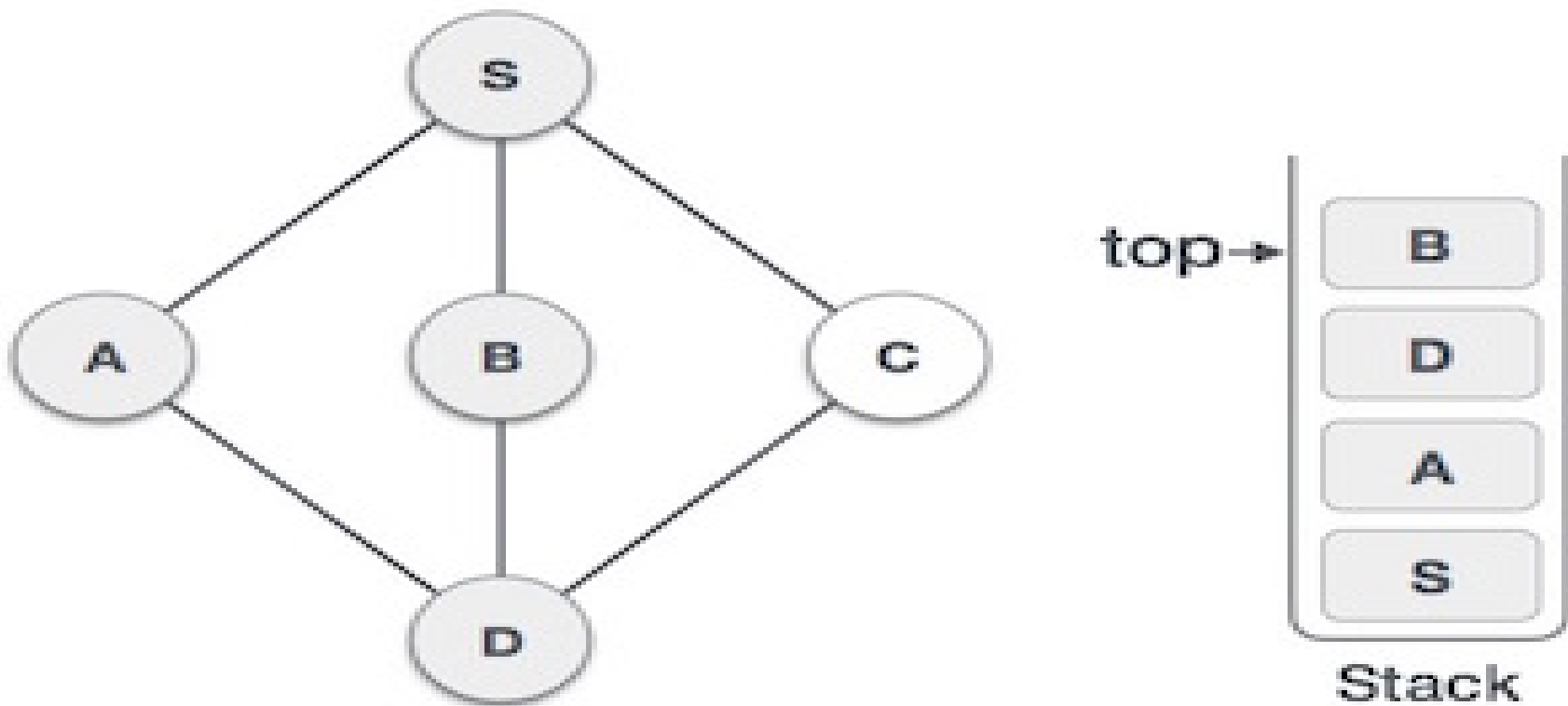
Mark A as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both S and D are adjacent to A but we are concerned for unvisited nodes only.



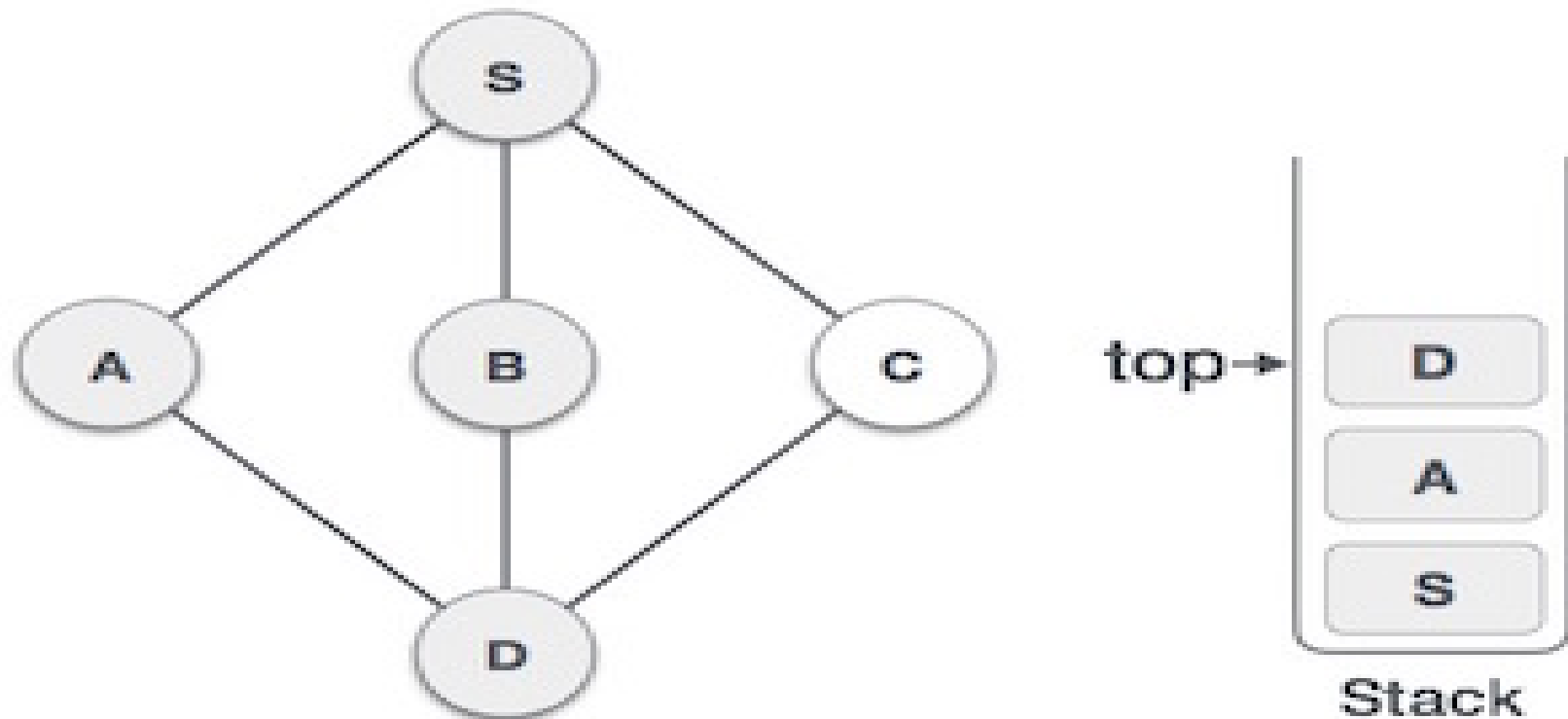
Visit D and mark it as visited and put onto the stack. Here, we have B and C nodes, which are adjacent to D and both are unvisited. However, we shall again choose in an alphabetical order.



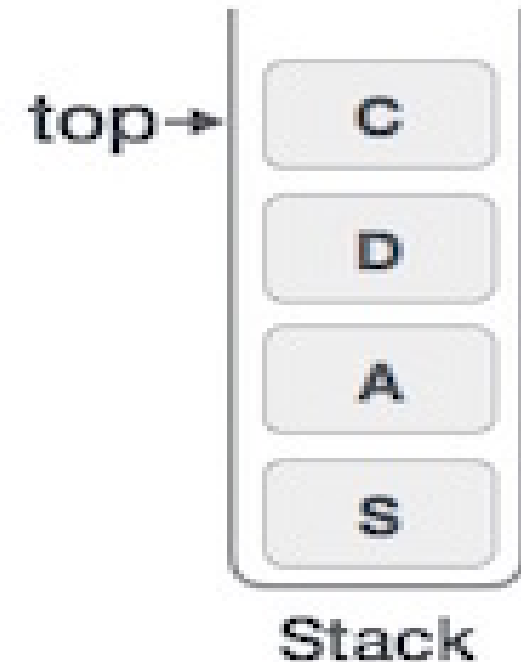
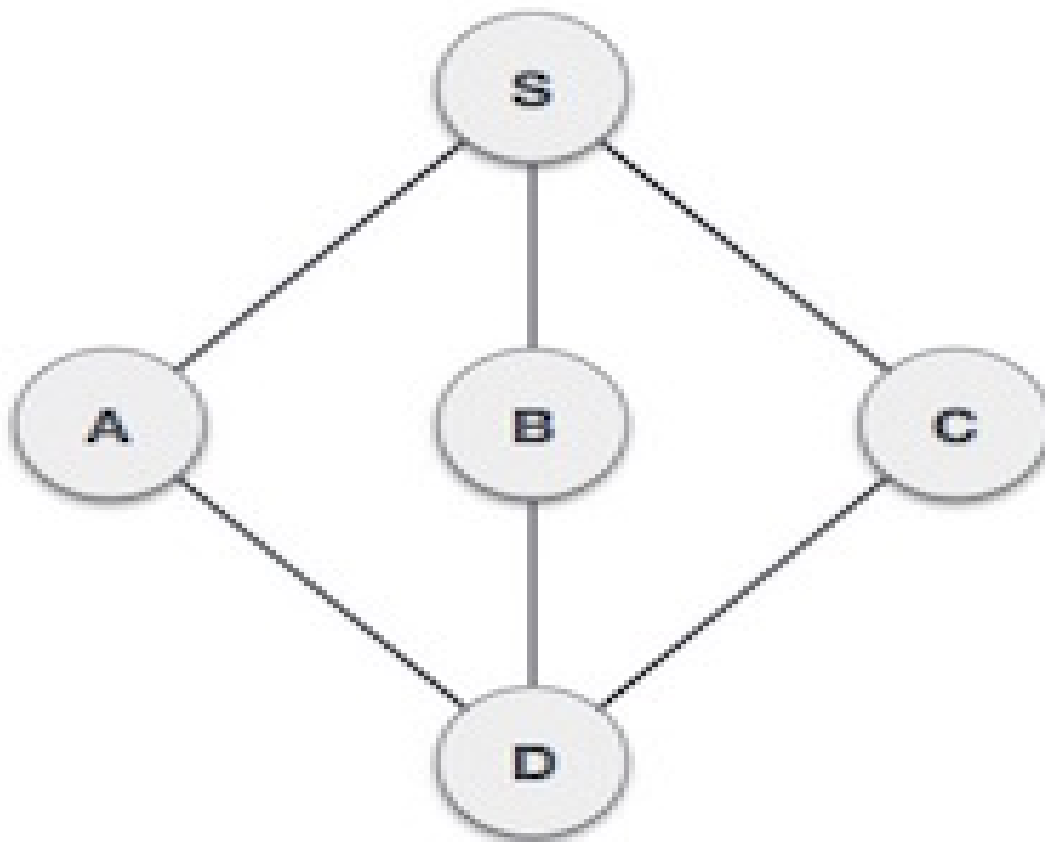
We choose B, mark it as visited and put onto the stack. Here B does not have any unvisited adjacent node. So, we pop B from the stack.



We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find D to be on the top of the stack.



Only unvisited adjacent node is from D is C now. So we visit C, mark it as visited and put it onto the stack.

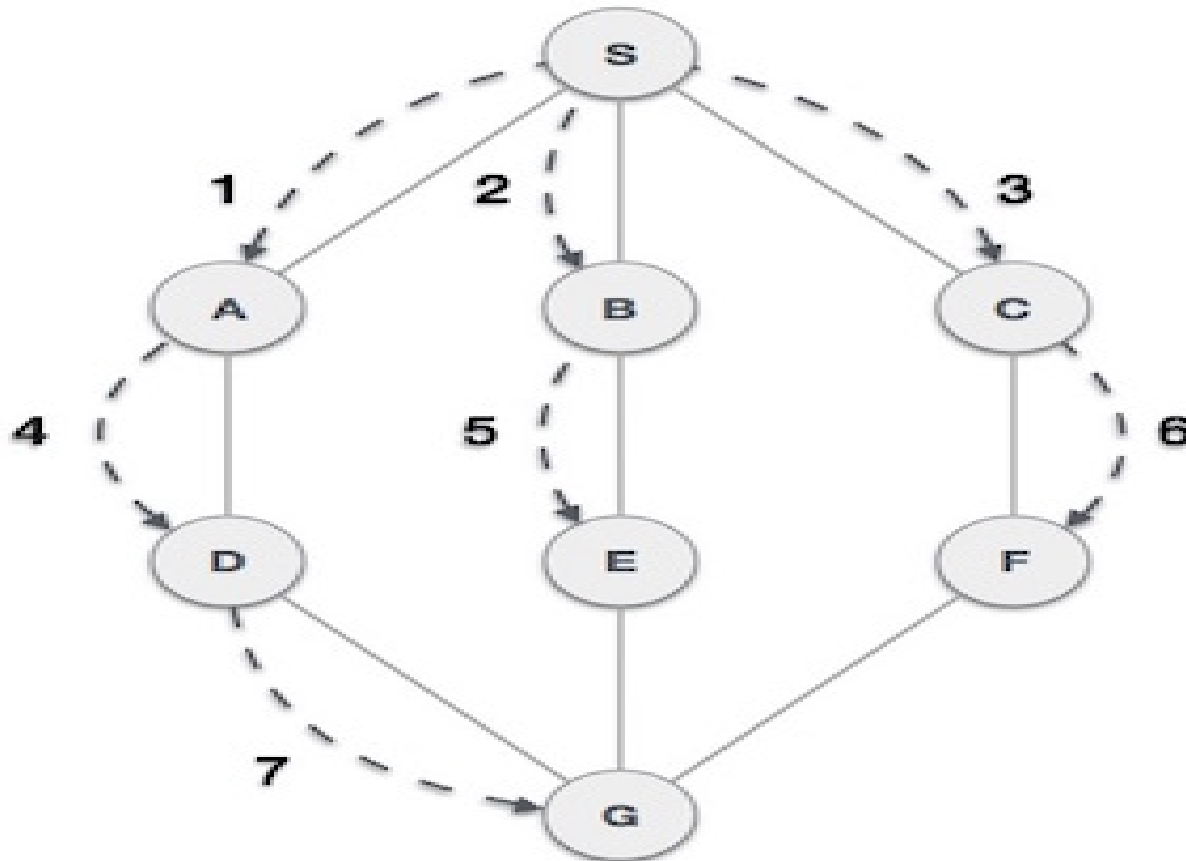


As C does not have any unvisited adjacent node so we keep popping the stack until we find a node that has an unvisited adjacent node. In this case, there's none and we keep popping until the stack is empty.

Thank You

Breadth First Search

Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

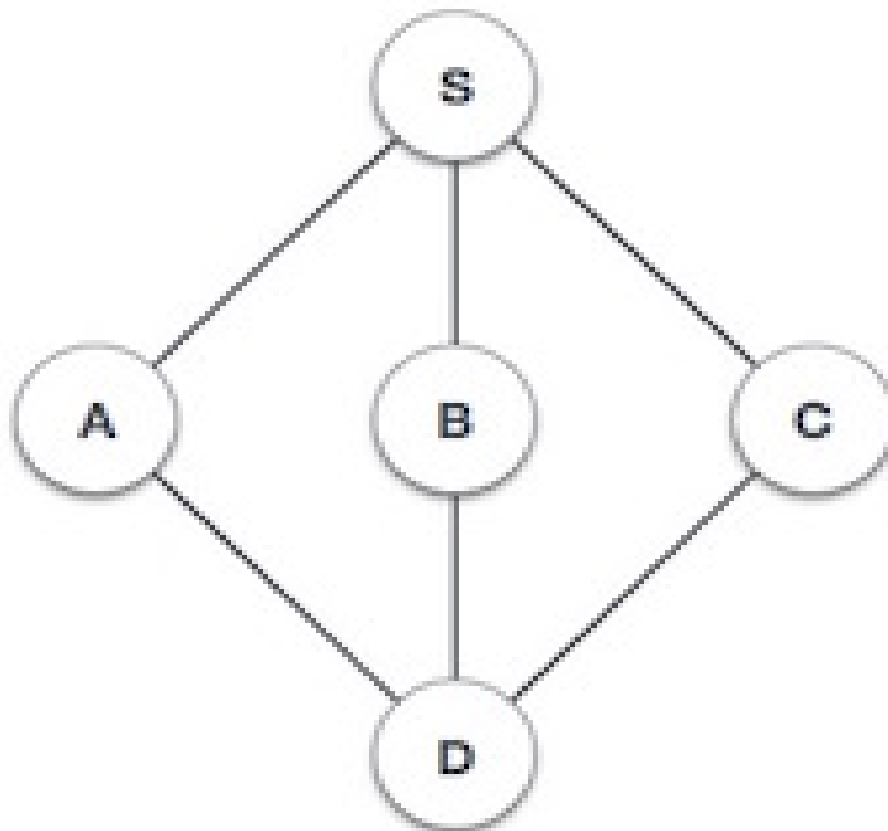
Rule 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.

Rule 2 – If no adjacent vertex is found, remove the first vertex from the queue.

Rule 3 – Repeat Rule 1 and Rule 2 until the queue is empty.

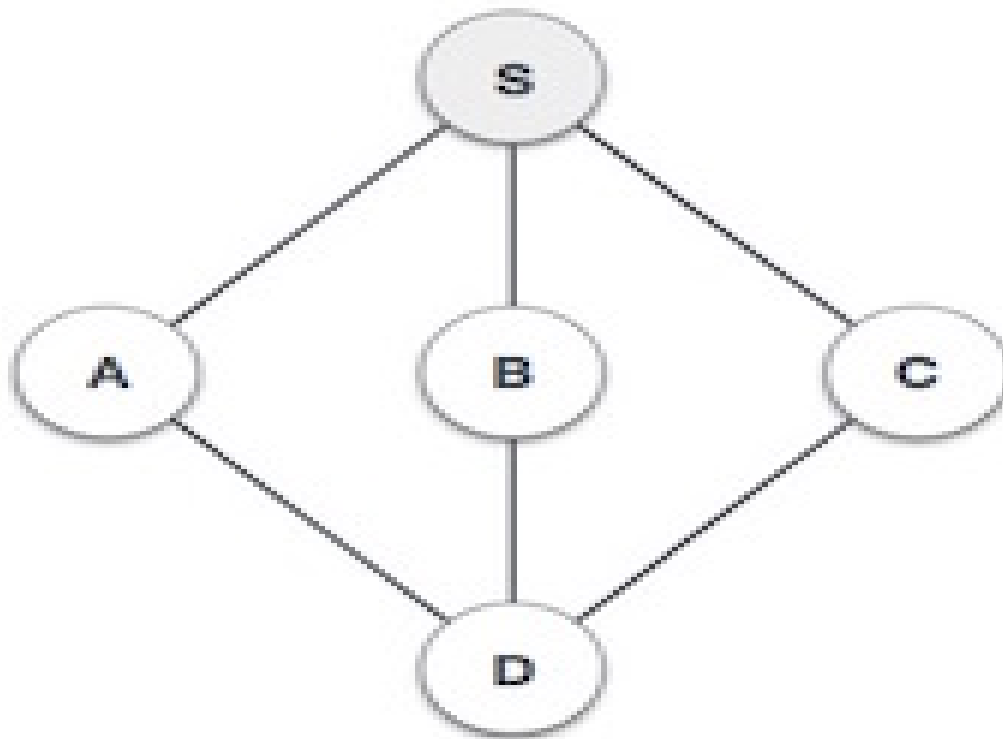
Example

Initialize the queue.



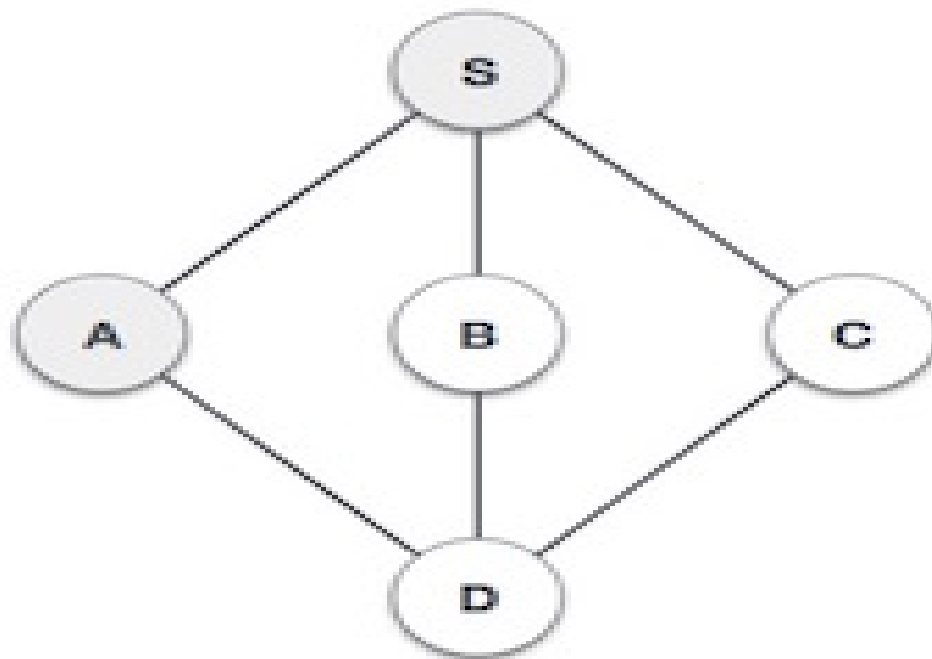
Queue

We start from visiting S (starting node), and mark it as visited.

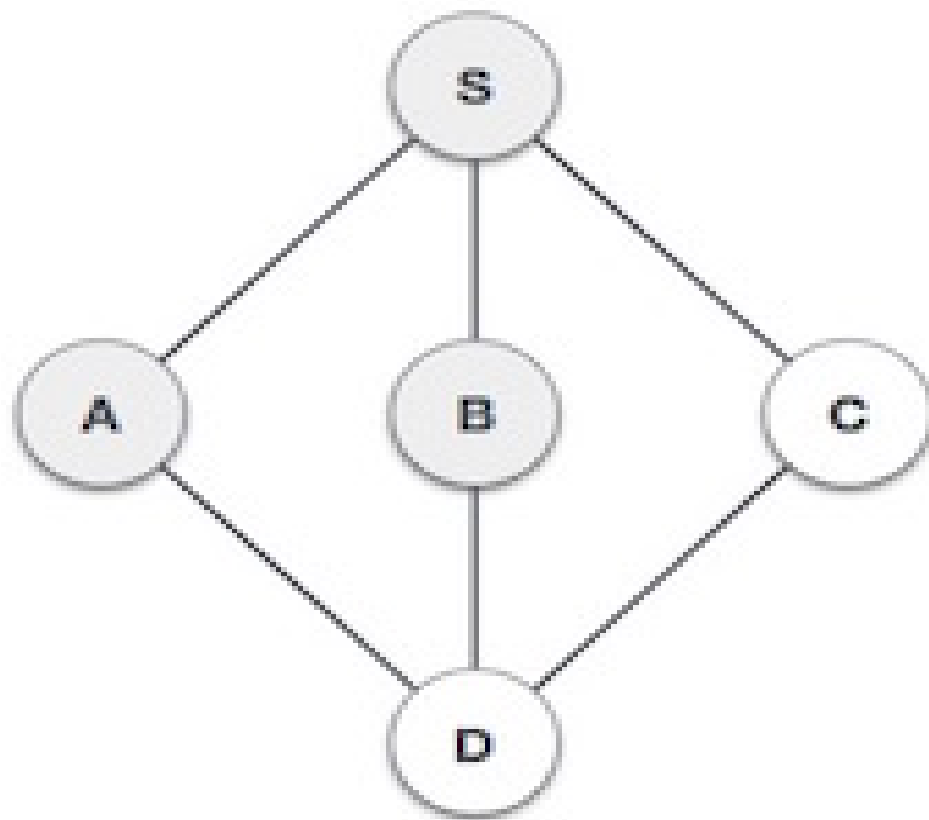


Queue

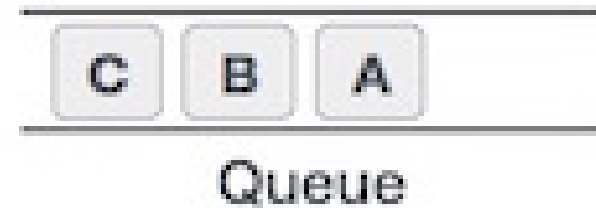
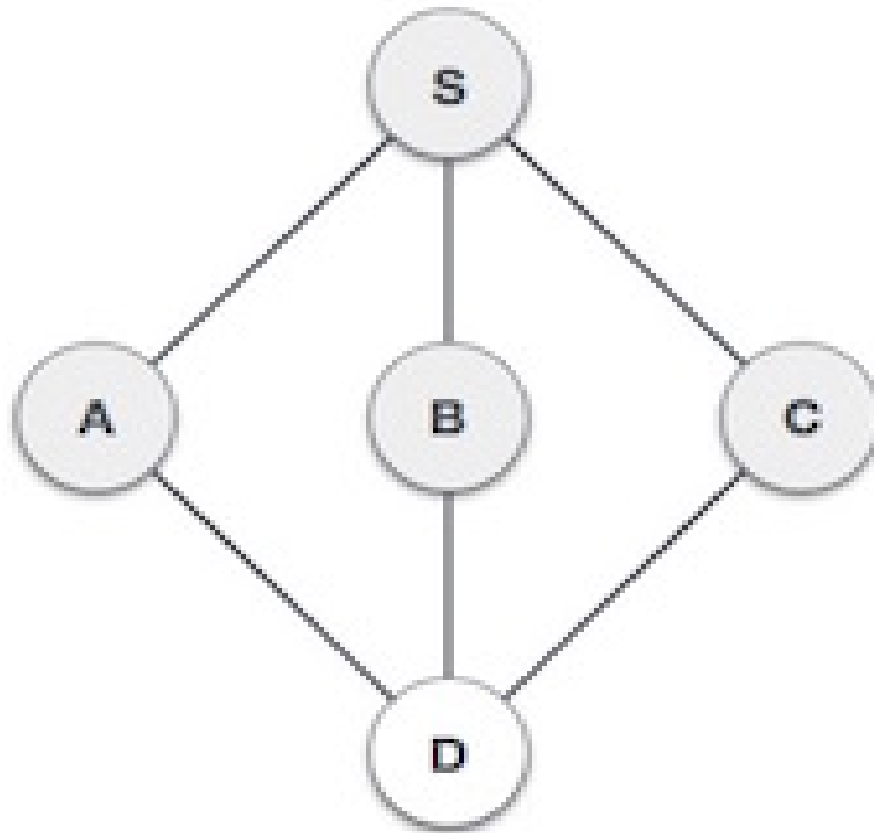
We then see an unvisited adjacent node from S. In this example, we have three nodes but alphabetically we choose A, mark it as visited and enqueue it.



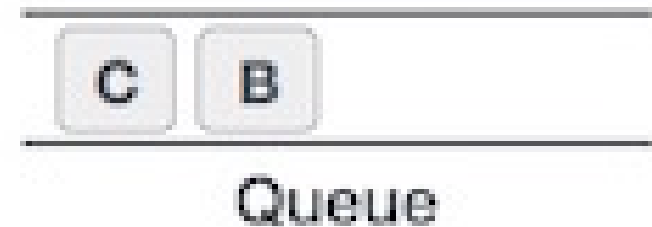
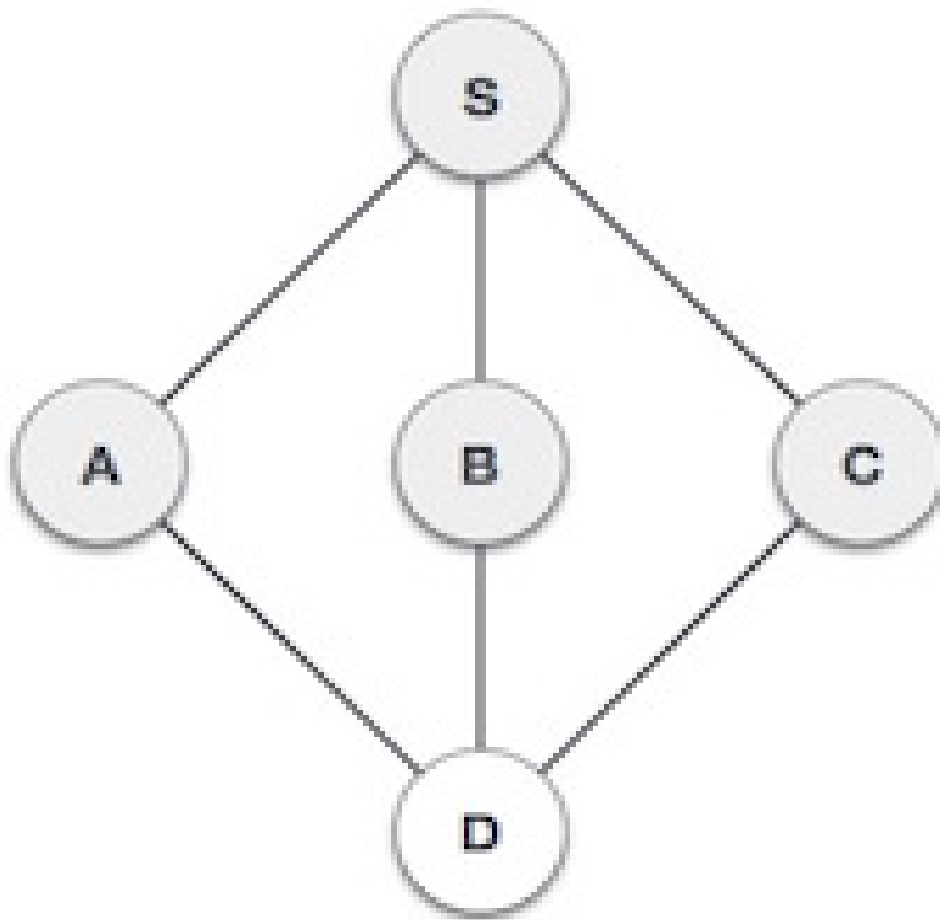
Next, the unvisited adjacent node from S is B. We mark it as visited and enqueue it.



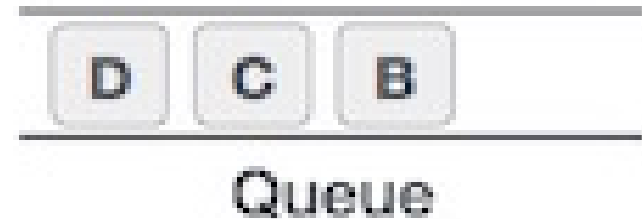
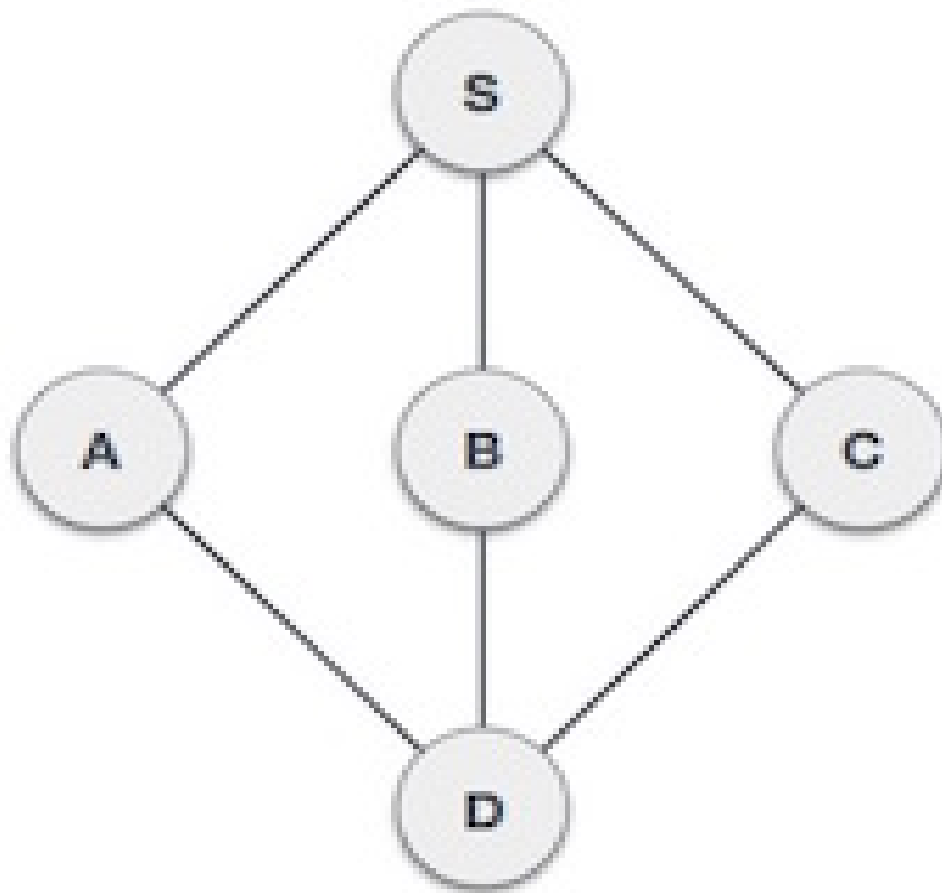
Next, the unvisited adjacent node from S is C. We mark it as visited and enqueue it.



Now, S is left with no unvisited adjacent nodes. So, we dequeue and find A.



From A we have D as unvisited adjacent node. We mark it as visited and enqueue it.

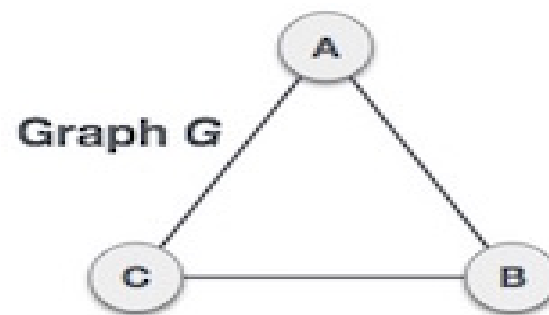


At this stage, we are left with no unmarked (unvisited) nodes. But as per the algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied, the program is over.

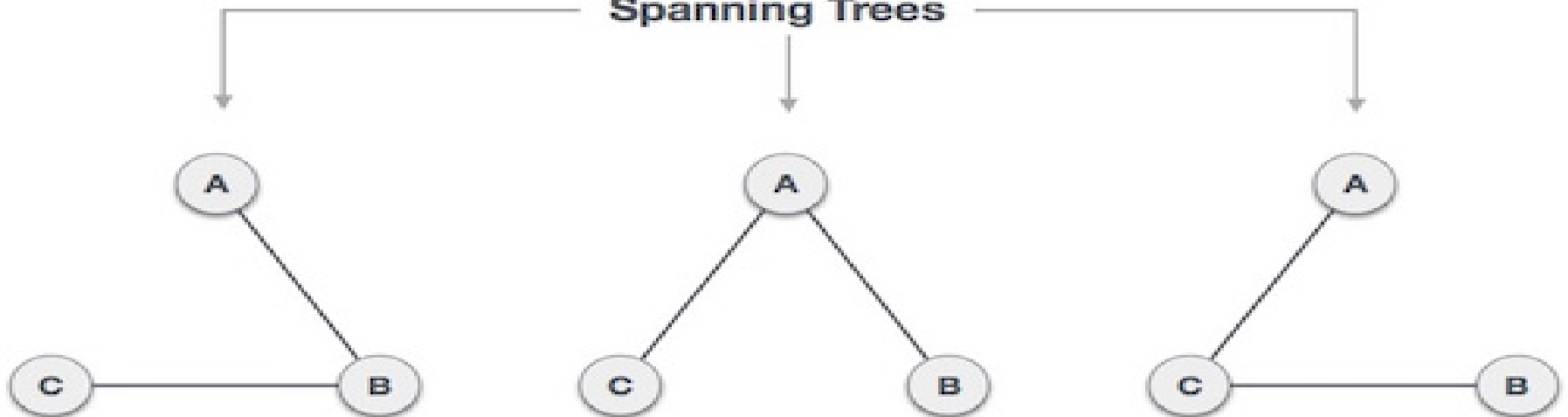
Spanning Tree

A spanning tree is a subset of Graph G , which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..

By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.



Spanning Trees



We found three spanning trees off one complete graph. A complete undirected graph can have maximum n^{n-2} number of spanning trees, where n is the number of nodes. In the above addressed example, n is 3, hence $3^{3-2} = 3$ spanning trees are possible.

General Properties of Spanning Tree:

We now understand that one graph can have more than one spanning tree. Following are a few properties of the spanning tree connected to graph G –

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G , have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is minimally connected.
- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is maximally acyclic.

Mathematical Properties of Spanning Tree:

Spanning tree has $n-1$ edges, where n is the number of nodes (vertices).

From a complete graph, by removing maximum $e - n + 1$ edges, we can construct a spanning tree.

A complete graph can have maximum n^{n-2} number of spanning trees.

Thus, we can conclude that spanning trees are a subset of connected Graph G and disconnected graphs do not have spanning tree.

Application of Spanning Tree:

Spanning tree is basically used to find a minimum path to connect all nodes in a graph. Common application of spanning trees are –

- Civil Network Planning
- Computer Network Routing Protocol
- Cluster Analysis

Let us understand this through a small example. Consider, city network as a huge graph and now plans to deploy telephone lines in such a way that in minimum lines we can connect to all city nodes. This is where the spanning tree comes into picture.

Minimum Spanning Tree (MST):

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph. In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

Minimum Spanning-Tree Algorithm:

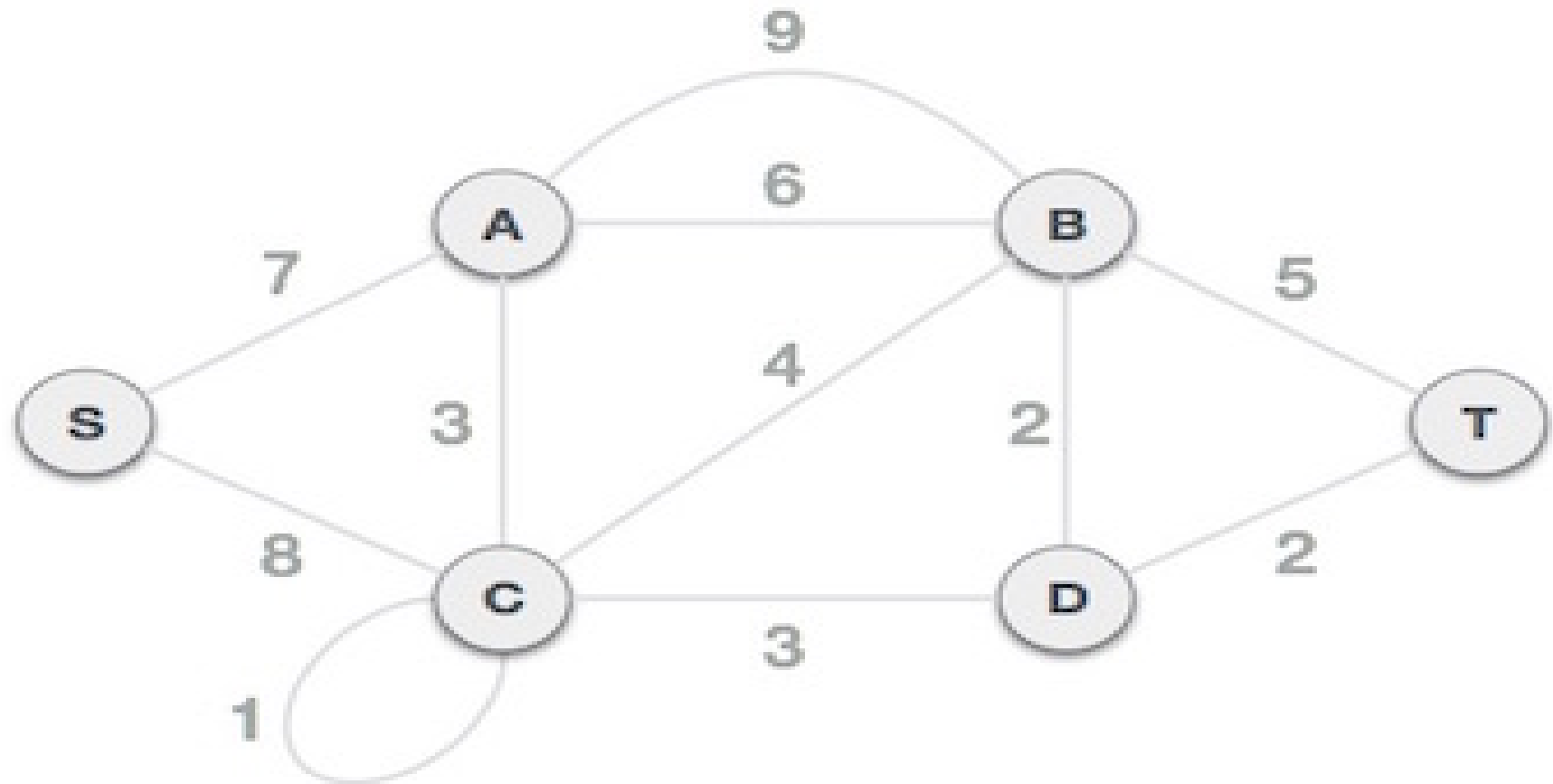
We shall learn about two most important spanning tree algorithms here –

- Kruskal's Algorithm
- Prim's Algorithm

Kruskal's Spanning Tree Algorithm

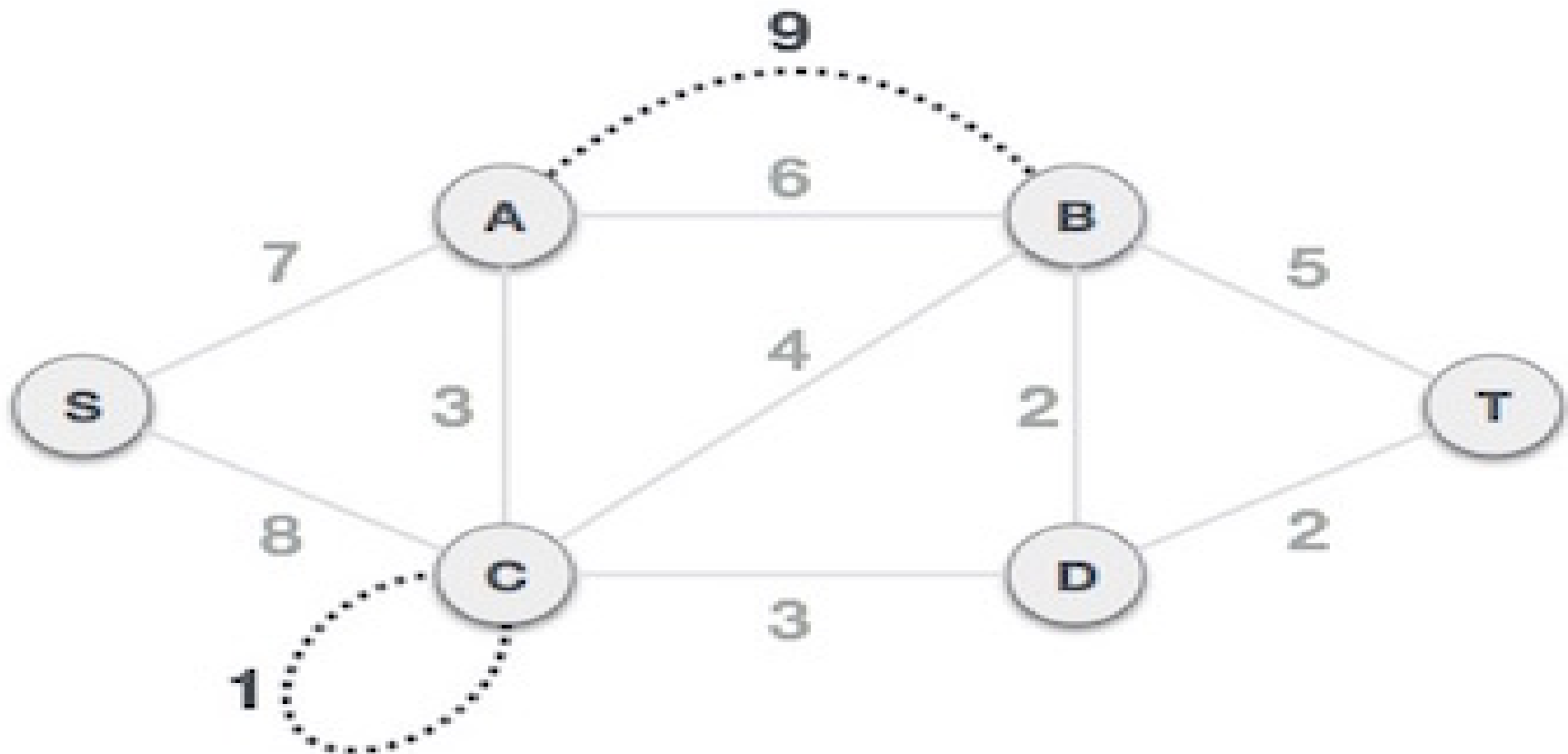
Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. This algorithm treats the graph as a forest and every node it has as an individual tree. A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.

To understand Kruskal's algorithm let us consider the following example –

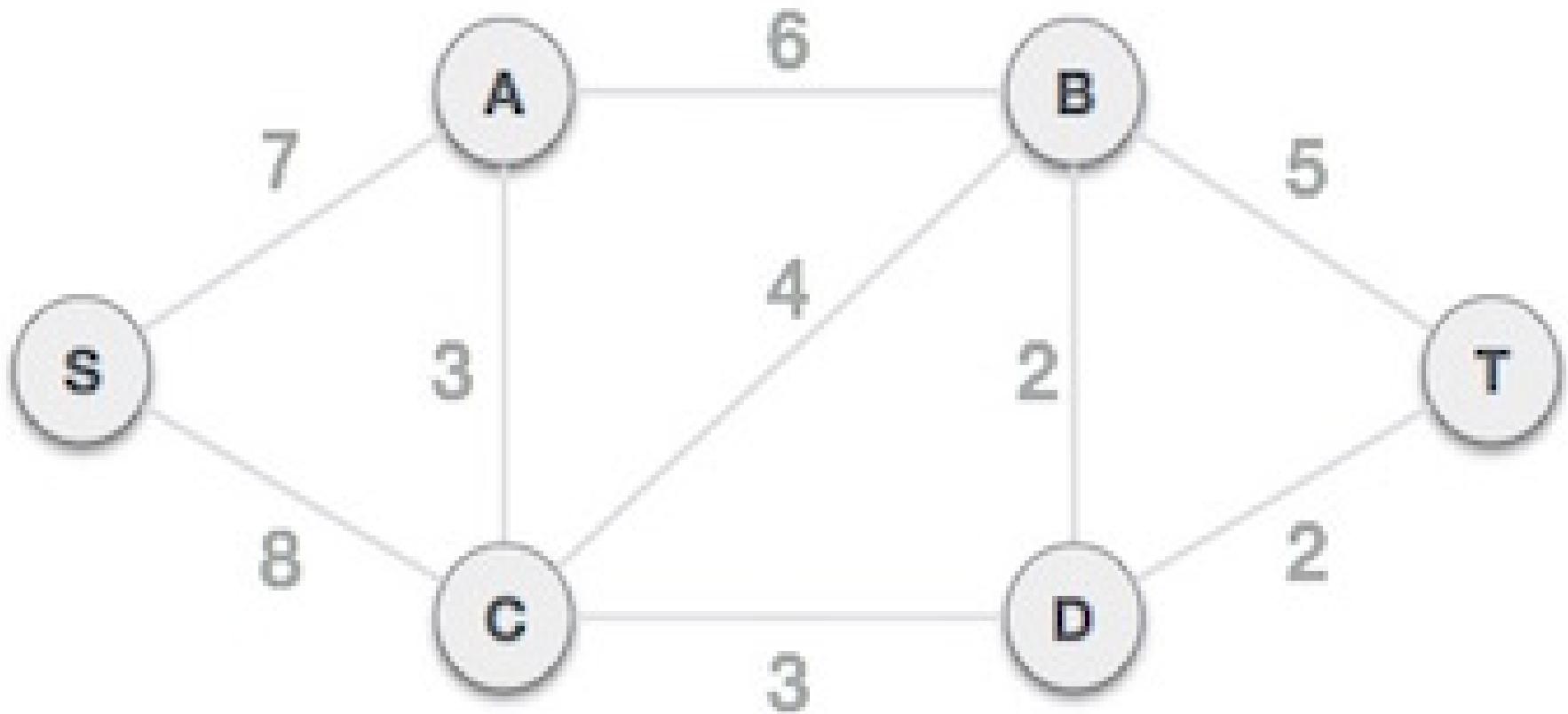


Step 1 - Remove all loops and Parallel Edges

Remove all loops and parallel edges from the given graph.



In case of parallel edges, keep the one which has the least cost associated and remove all others.



Step 2 - Arrange all edges in their increasing order of weight

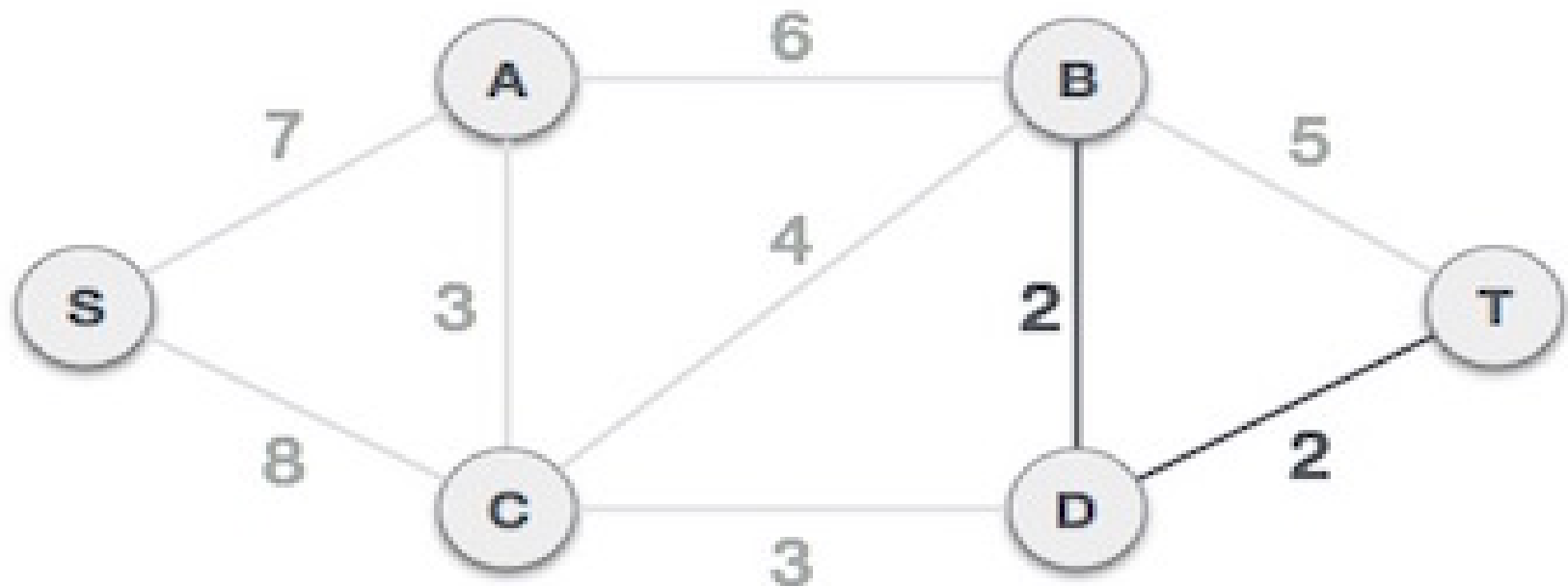
The next step is to create a set of edges and weight, and arrange them in an ascending order of weightage (cost).

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

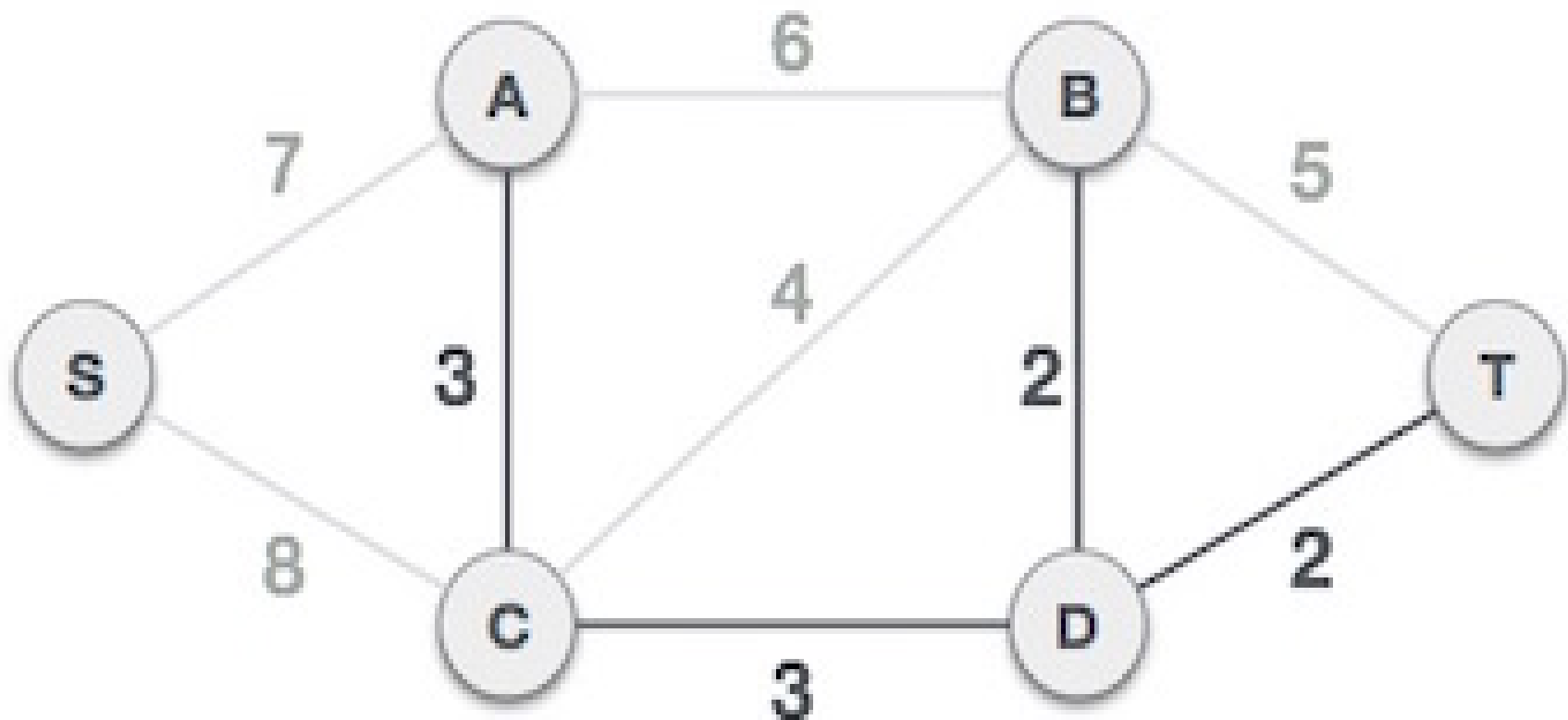
Step 3 - Add the edge which has the least weightage

Now we start adding edges to the graph beginning from the one which has the least weight. Throughout, we shall keep checking that the spanning properties remain intact. In case, by adding one edge, the spanning tree property does not hold then we shall consider not to include the edge in the graph.

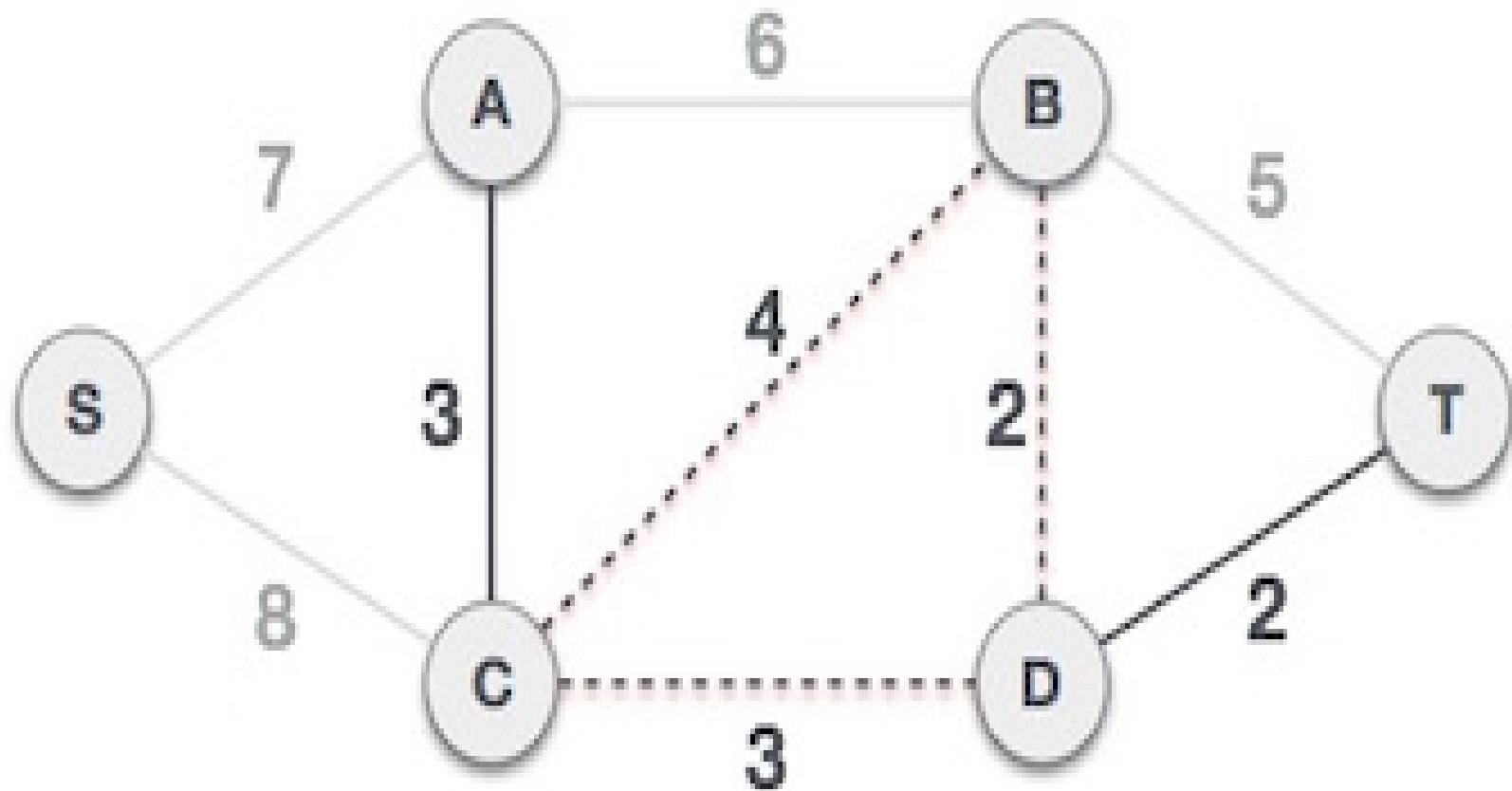
The least cost is 2 and edges involved are B,D and D,T. We add them. Adding them does not violate spanning tree properties, so we continue to our next edge selection.



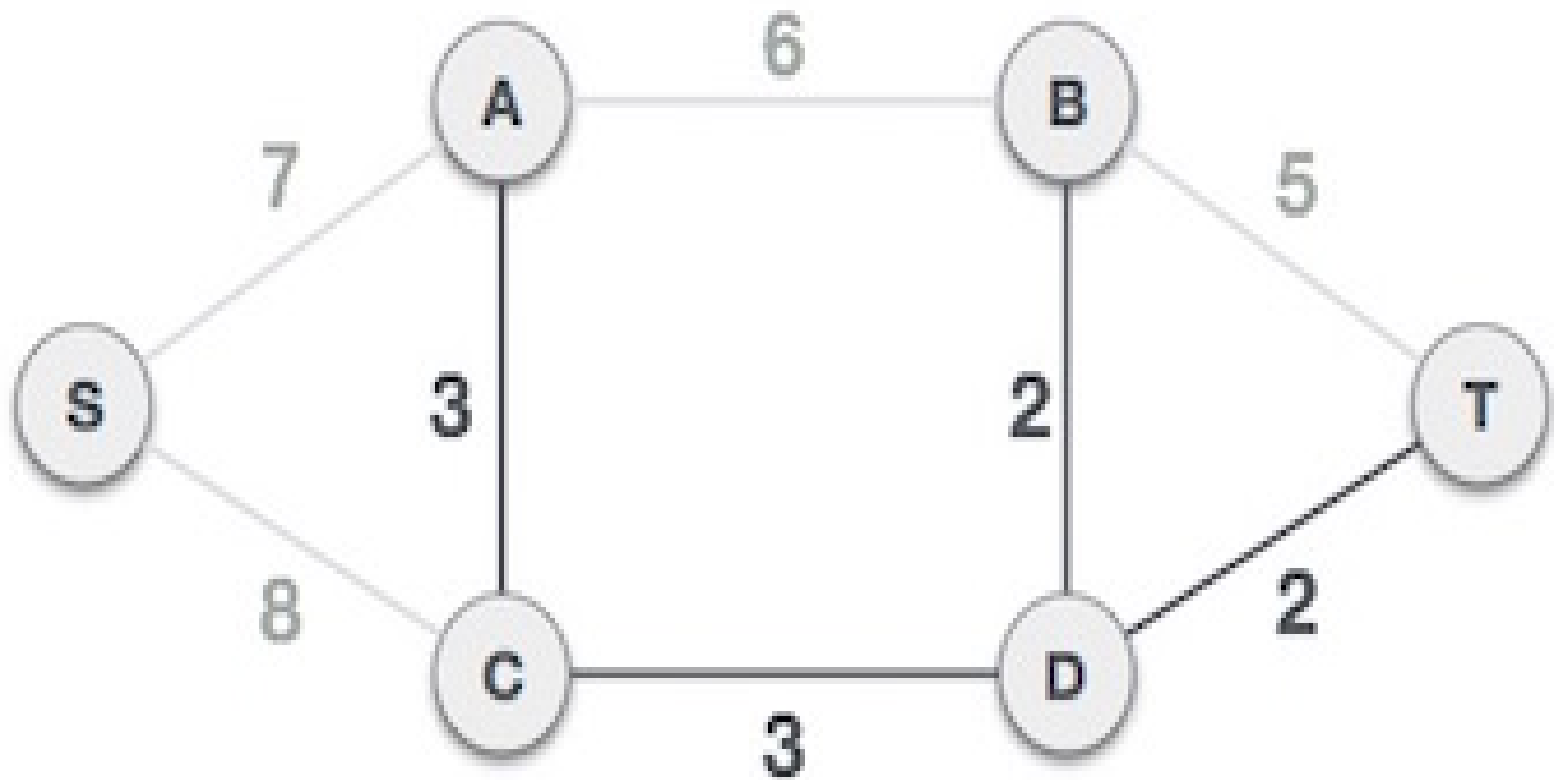
Next cost is 3, and associated edges are A,C and C,D. We add them again –



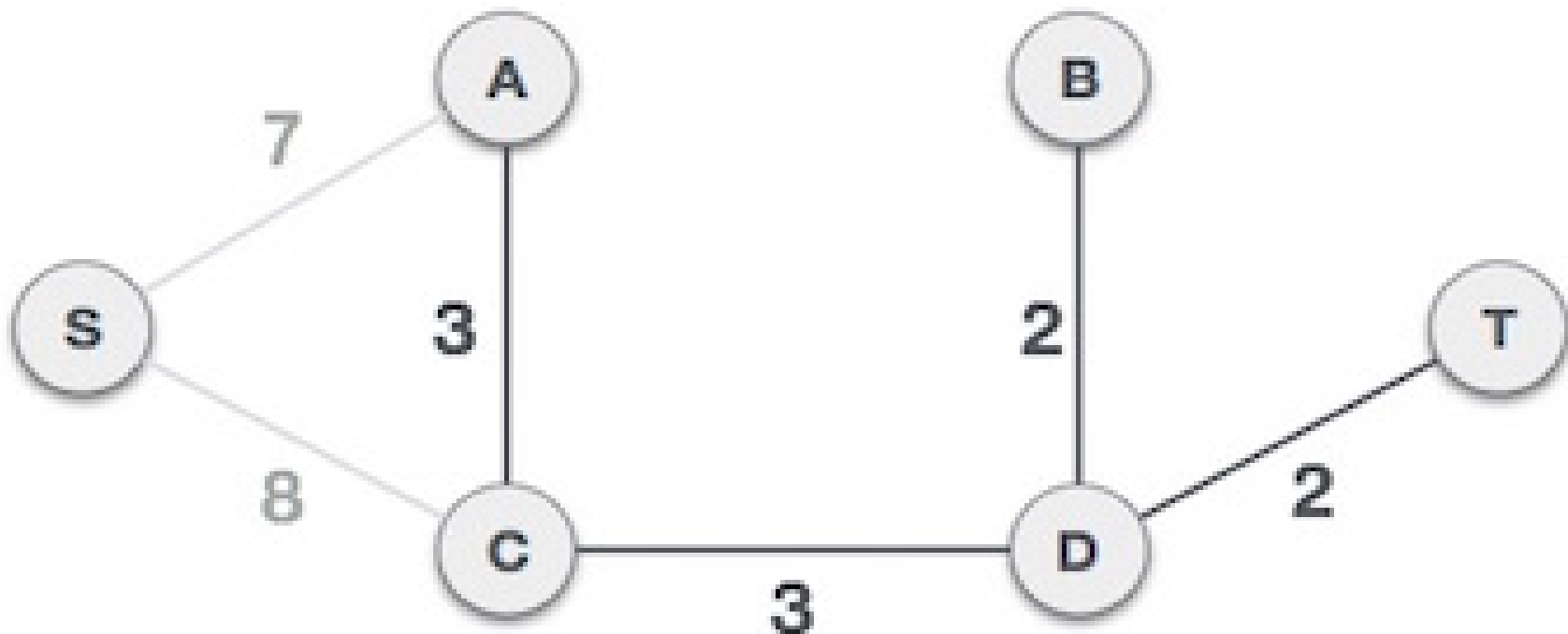
Next cost in the table is 4, and we observe that adding it will create a circuit in the graph. –



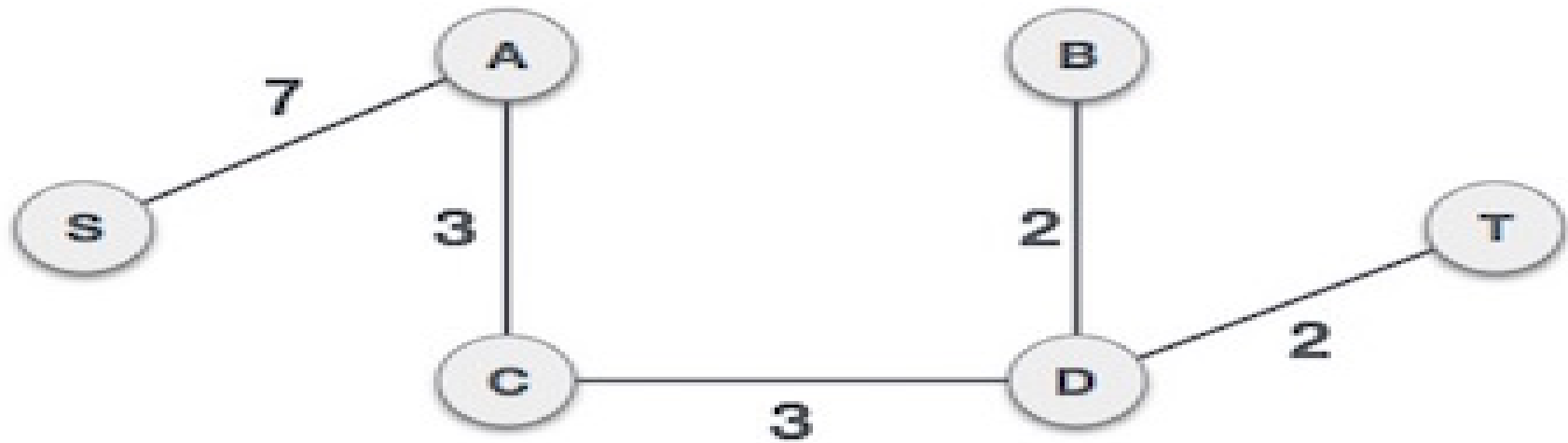
We ignore it. In the process we shall ignore/avoid all edges that create a circuit.



We observe that edges with cost 5 and 6 also create circuits. We ignore them and move on.



Now we are left with only one node to be added. Between the two least cost edges available 7 and 8, we shall add the edge with cost 7.



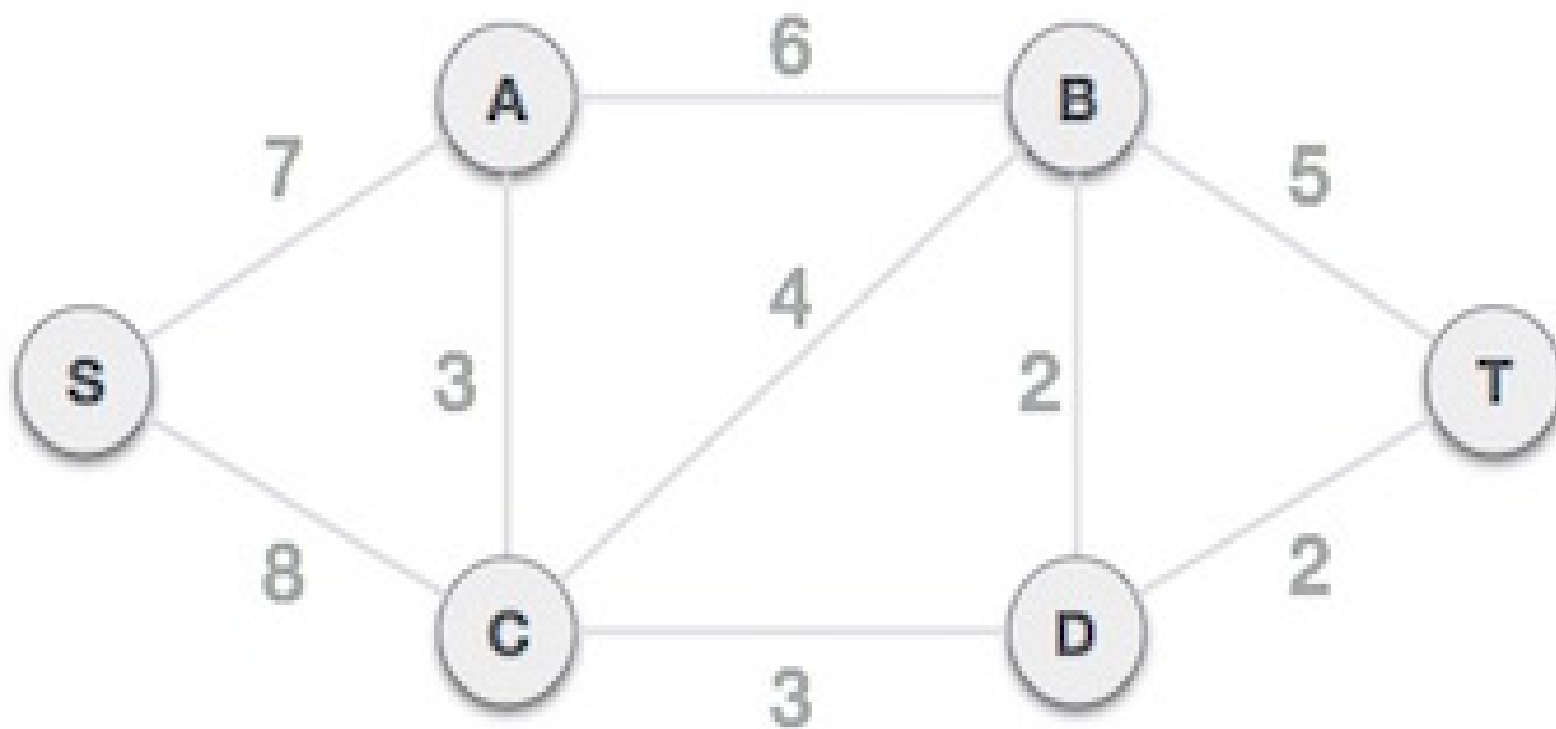
By adding edge S,A we have included all the nodes of the graph and we now have minimum cost spanning tree.

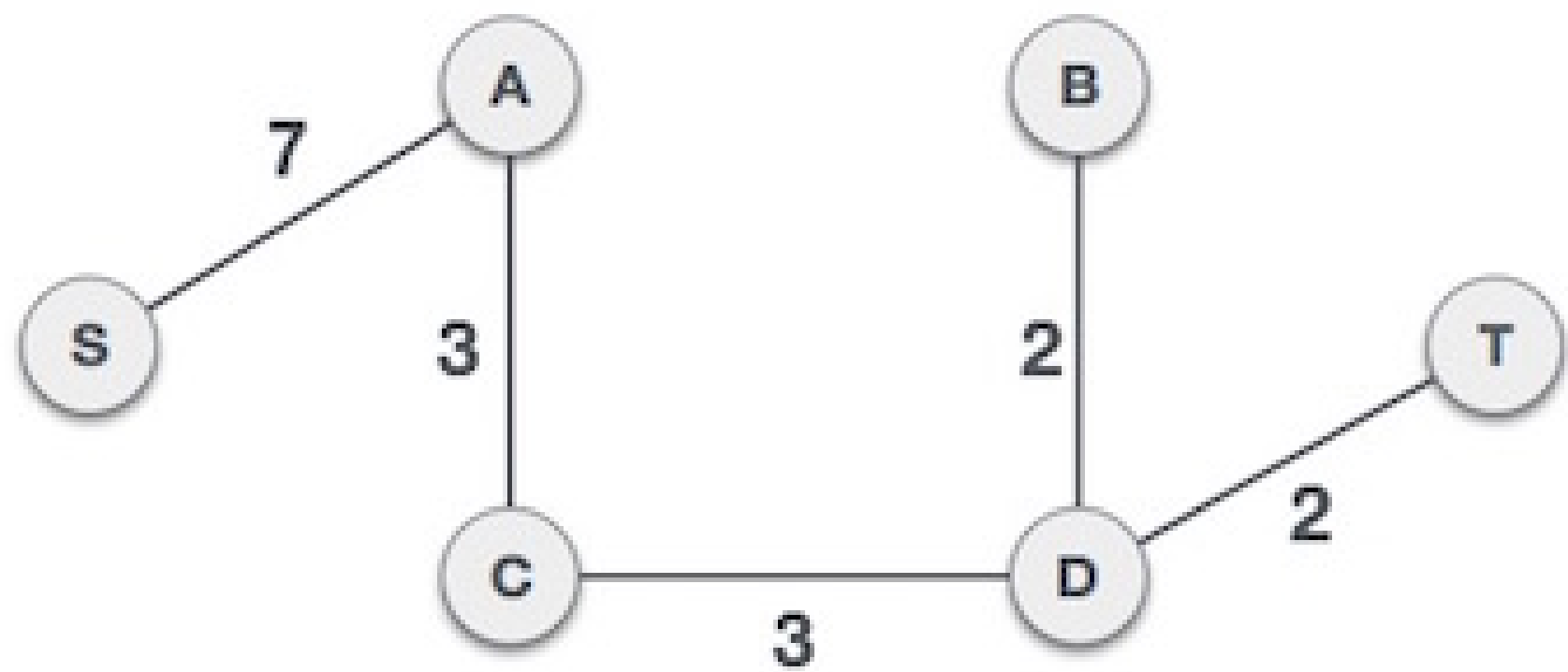
Prim's Spanning Tree Algorithm

Prim's algorithm to find minimum cost spanning tree (as Kruskal's algorithm) uses the greedy approach. Prim's algorithm shares a similarity with the shortest path first algorithms.

Prim's algorithm, in contrast with Kruskal's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.

Example





Hashing

Hash Table:

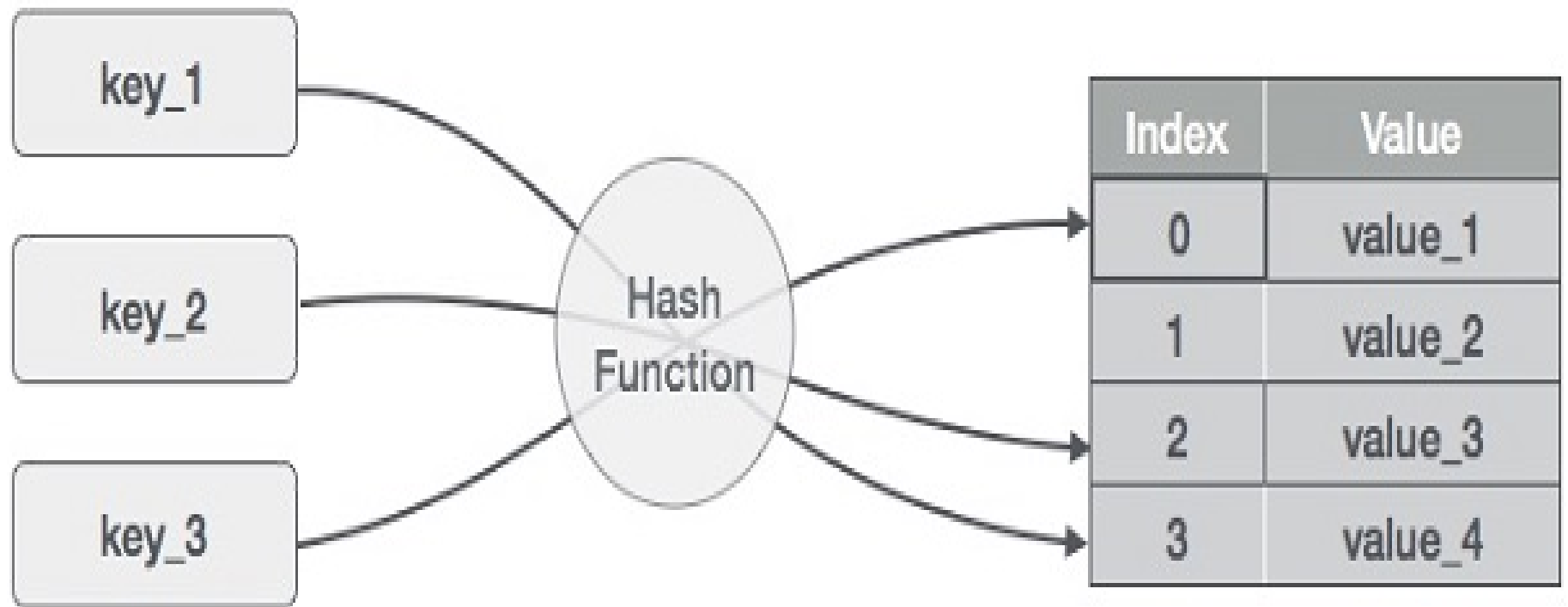
Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.

Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

Hashing:

Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Consider an example of hash table of size 20, and the following items are to be stored.

Item are in the (key,value) format.



(1,20)

(2,70)

(42,80)

(4,25)

(12,44)

(14,32)

(17,11)

(13,78)

(37,98)

Sr.No.	Key	Hash	ArrayIndex
1	1	$1 \% 20 = 1$	1
2	2	$2 \% 20 = 2$	2
3	42	$42 \% 20 = 2$	2
4	4	$4 \% 20 = 4$	4
5	12	$12 \% 20 = 12$	12
6	14	$14 \% 20 = 14$	14
7	17	$17 \% 20 = 17$	17
8	13	$13 \% 20 = 13$	13
9	37	$37 \% 20 = 17$	17

Linear Probing:

As we can see, it may happen that the hashing technique is used to create an already used index of the array. In such a case, we can search the next empty location in the array by looking into the next cell until we find an empty cell. This technique is called linear probing.

Basic Operations:

Following are the basic primary operations of a hash table.

Search – Searches an element in a hash table.

Insert – inserts an element in a hash table.

delete – Deletes an element from a hash table.

Hash function:

Hash function is a function which is applied on a key by which it produces an integer, which can be used as an address of hash table. Hence one can use the same hash function for accessing the data from the hash table. In this the integer returned by the hash function is called hash key.

Types of hash function

There are various types of hash function which are used to place the data in a hash table,

1. Division method
2. Mid square method
3. Digit folding method

In this the hash function is dependent upon the remainder of a division. For example:-if the record 52,68,99,84 is to be placed in a hash table and let us take the table size is 10.

Then:

$h(\text{key}) = \text{record} \% \text{table size.}$

$$2 = 52 \% 10$$

$$8 = 68 \% 10$$

$$9 = 99 \% 10$$

$$4 = 84 \% 10$$

Division method

2. Mid square method:

In this method firstly key is squared and then mid part of the result is taken as the index. For example: consider that if we want to place a record of 3101 and the size of table is 1000. So $3101 * 3101 = 9616201$ i.e. $h(3101) = 162$ (middle 3 digit)

3. Digit folding method:

In this method the key is divided into separate parts and by using some simple operations these parts are combined to produce a hash key. For example: consider a record of 12465512 then it will be divided into parts i.e. 124, 655, 12. After dividing the parts combine these parts by adding it.

$$\begin{aligned} H(\text{key}) &= 124 + 655 + 12 \\ &= 791 \end{aligned}$$

Thank You