



# UNIVERSITY OF ENGINEERING & MANAGEMENT, KOLKATA

Course Name : Design Analysis and Algorithm Laboratory



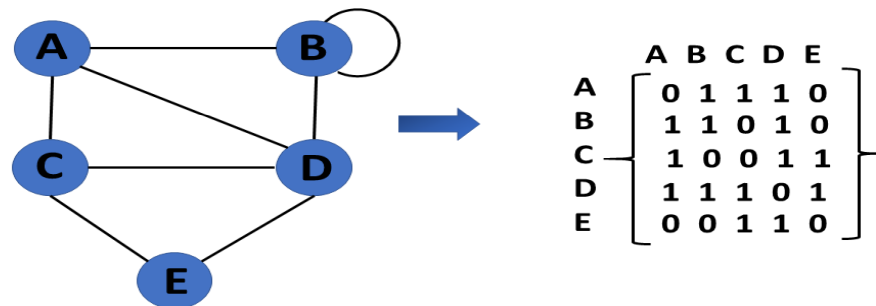
# Adjacency matrix

A sequential representation is an adjacency matrix.

It's used to show which nodes are next to one another. I.e., is there any connection between nodes in a graph?

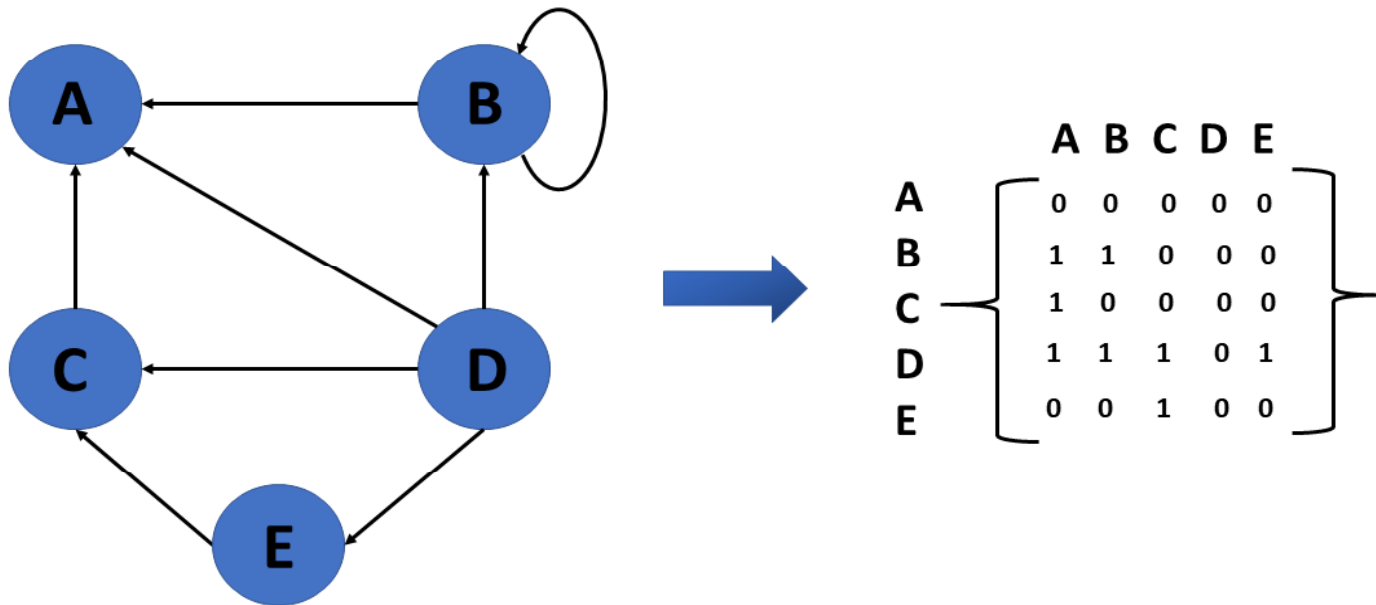
You create an  $M \times M$  matrix  $G$  for this representation. If an edge exists between vertex  $a$  and vertex  $b$ , the corresponding element of  $G$ ,  $g_{i,j} = 1$ , otherwise  $g_{i,j} = 0$ . If there is a weighted graph, you can record the edge's weight instead of 1s and 0s.

Undirected Graph Representation



# Adjacency matrix

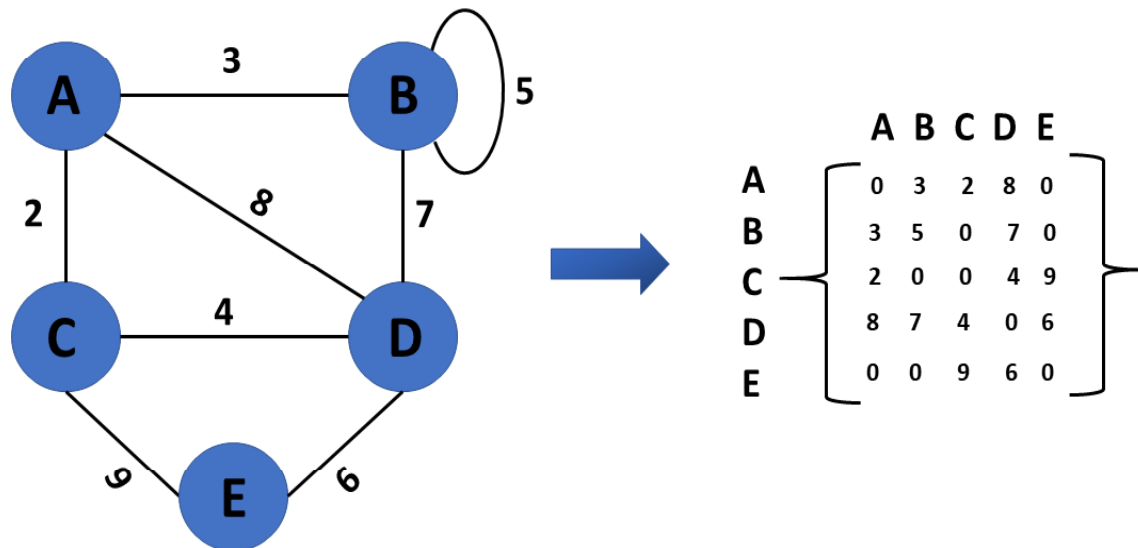
Directed Graph Representation



# Adjacency matrix

Weighted Undirected Graph Representation

Weight or cost is indicated at the graph's edge, a weighted graph representing these values in the matrix.



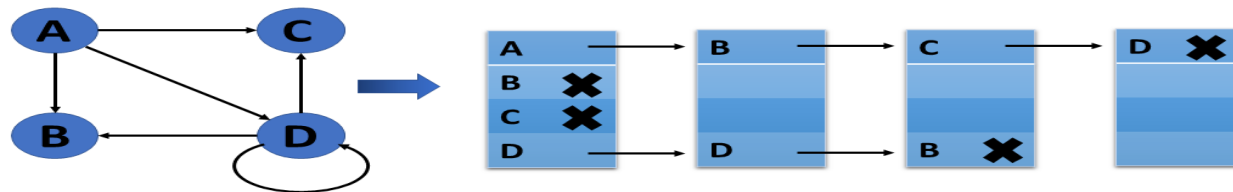
# Adjacency List

A linked representation is an adjacency list.

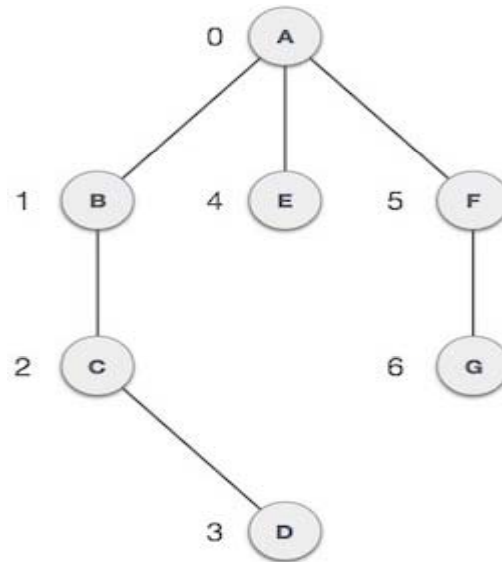
You keep a list of neighbors for each vertex in the graph in this representation. It means that each vertex in the graph has a list of its neighboring vertices.

You have an array of vertices indexed by the vertex number, and the corresponding array member for each vertex  $x$  points to a singly linked list of  $x$ 's neighbors.

Weighted Undirected Graph Representation Using Linked-List



# Graph Data Structure



## 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.

## Breadth First Search (BFS)

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.

## Algorithm for BFS:

Step 1: Choose any one node randomly, to start traversing.

Step 2: Visit its adjacent unvisited node.

Step 3: Mark it as visited in the boolean array and display it.

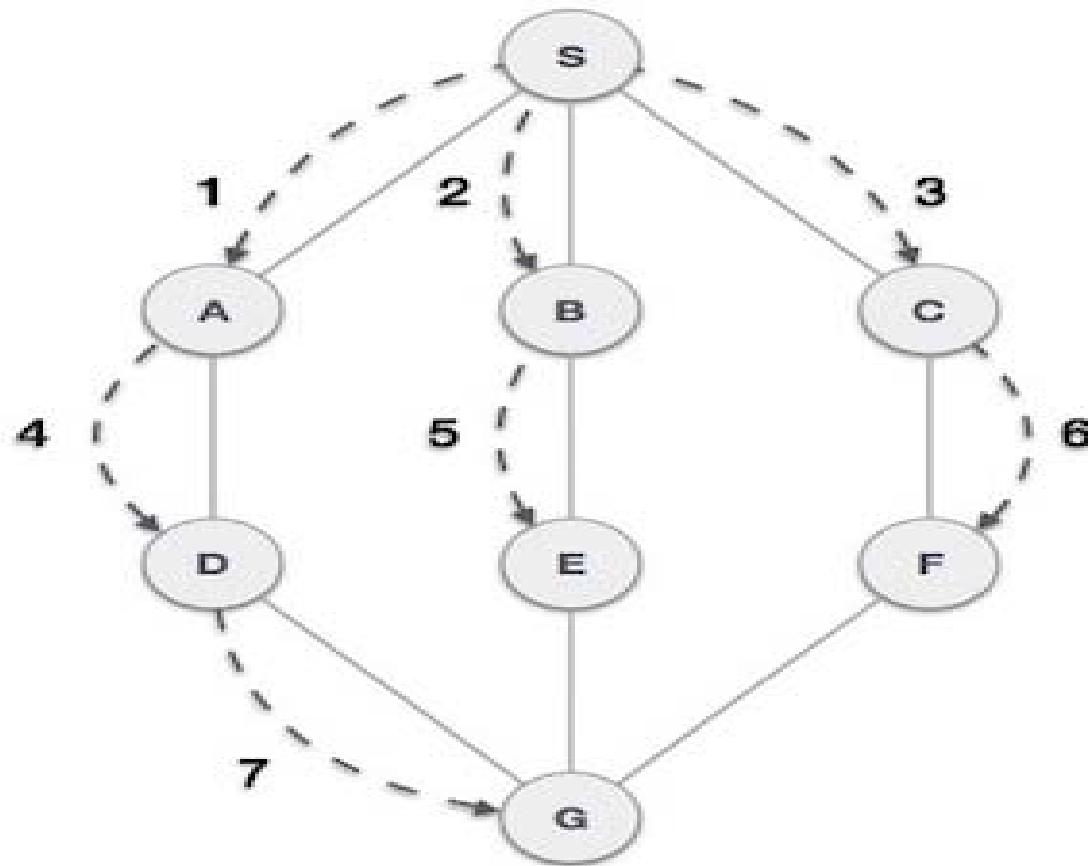
Step 4: Insert the visited node into the queue.

Step 5: If there is no adjacent node, remove the first node from the queue.

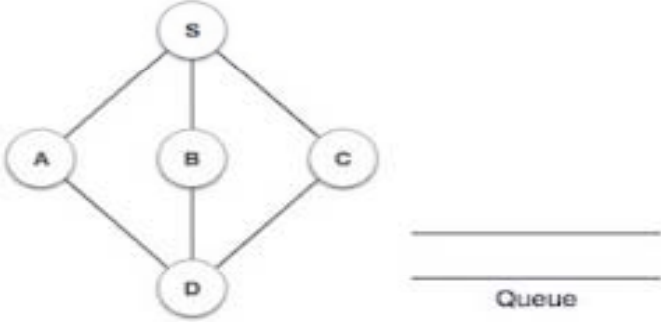
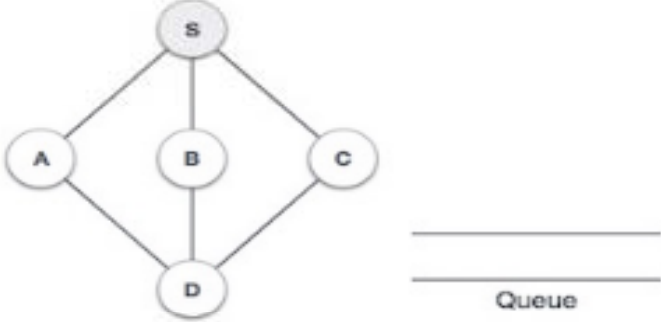
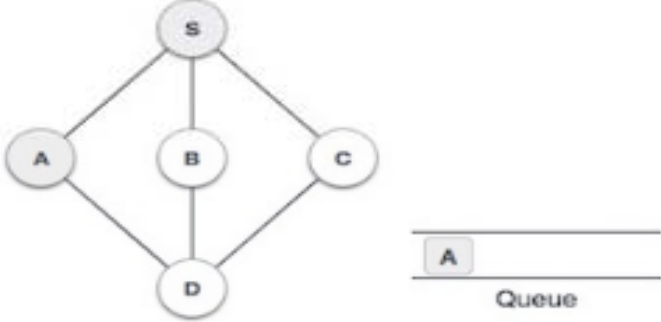
Step 6: Repeat the above steps until the queue is empty.



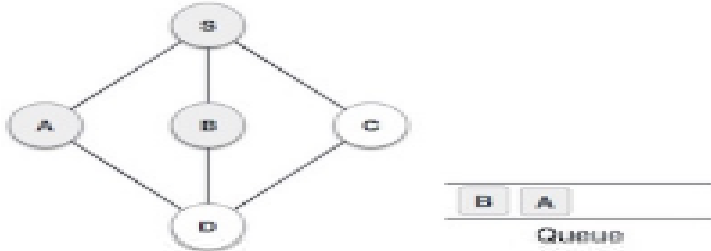
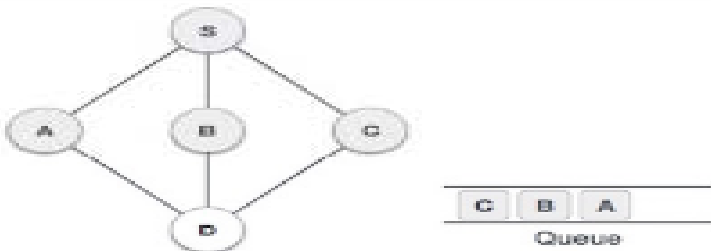
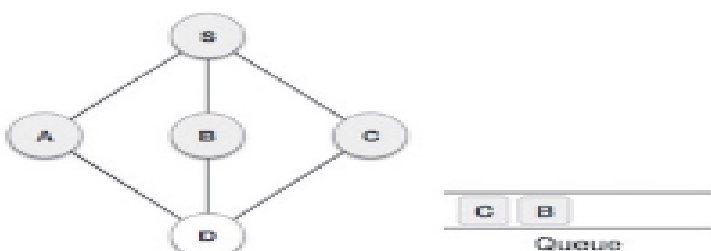
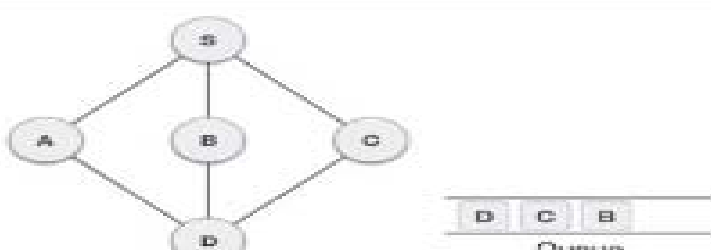
# Breadth First Search (BFS)



# Breadth First Search (BFS)

Step	Traversal	Description
1		Initialize the queue.
2		We start from visiting <b>S</b> (starting node), and mark it as visited.
3		We then see an unvisited adjacent node from <b>S</b> . In this example, we have three nodes but alphabetically we choose <b>A</b> , mark it as visited and enqueue it.

# Breadth First Search (BFS)

4	 <p>Graph diagram showing nodes S, A, B, C, and D. S is the source node. The queue contains nodes B and A.</p>	Next, the unvisited adjacent node from S is B. We mark it as visited and enqueue it.
5	 <p>Graph diagram showing nodes S, A, B, C, and D. S is the source node. The queue contains nodes C, B, and A.</p>	Next, the unvisited adjacent node from S is C. We mark it as visited and enqueue it.
6	 <p>Graph diagram showing nodes S, A, B, C, and D. S is the source node. The queue contains nodes C and B.</p>	Now, S is left with no unvisited adjacent nodes. So, we dequeue and find A.
7	 <p>Graph diagram showing nodes S, A, B, C, and D. S is the source node. The queue contains nodes D, C, and B.</p>	From A we have D as unvisited adjacent node. We mark it as visited and enqueue it.

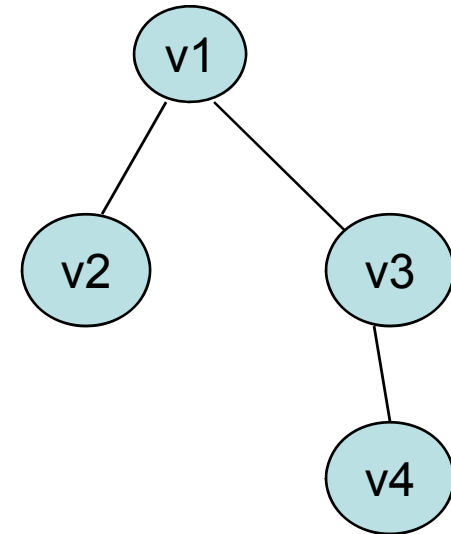
At this stage, we are left with no unmarked (unvisited) nodes. But as per the algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied, the program is over.

## Breadth First Search (BFS) Complexity

```
bfs(G,S)
{
    q=queue(); → O(1)
    q.enqueue(s);

    while(!q.empty())
    {
        v=q.dequeue(); // Dequeue the element and mark as visited

        for all u adjacent v
        {
            if(u is not visited)
            {
                q.enqueue(u); // Enqueue the element
            }
        }
    }
}
```



→  $O(E)$  [This loop will call  $E$  [number of edges] times]  
Each loop will repeat for  $V$  (Vertex) times  
→  $O(E+V)$

V1, V2, V3, V4

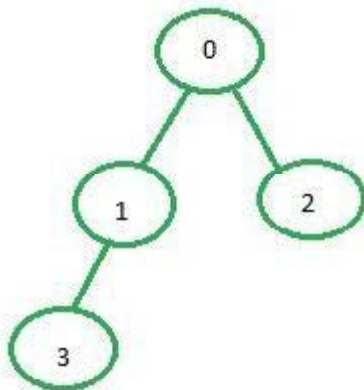
Visited

**Time complexity of BFS →  $O(E+V)$**

## Implementation of BFS using adjacency matrix

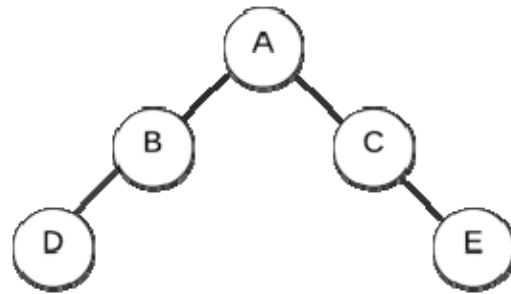
Breadth First Search (BFS) has been discussed in this article which uses adjacency list for the graph representation. In this article, adjacency matrix will be used to represent the graph.

Adjacency matrix representation: In adjacency matrix representation of a graph, the matrix  $mat[][]$  of size  $n*n$  (where  $n$  is the number of vertices) will represent the edges of the graph where  $mat[i][j] = 1$  represents that there is an edge between the vertices  $i$  and  $j$  while  $mat[i][j] = 0$  represents that there is no edge between the vertices  $i$  and  $j$ .



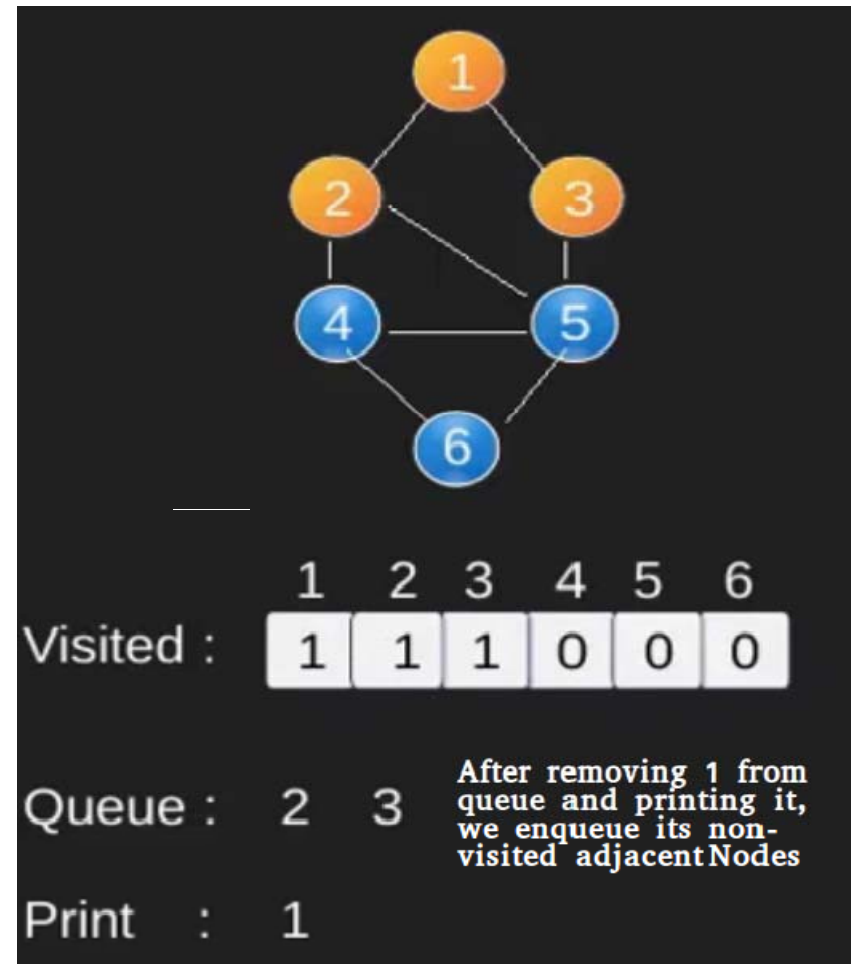
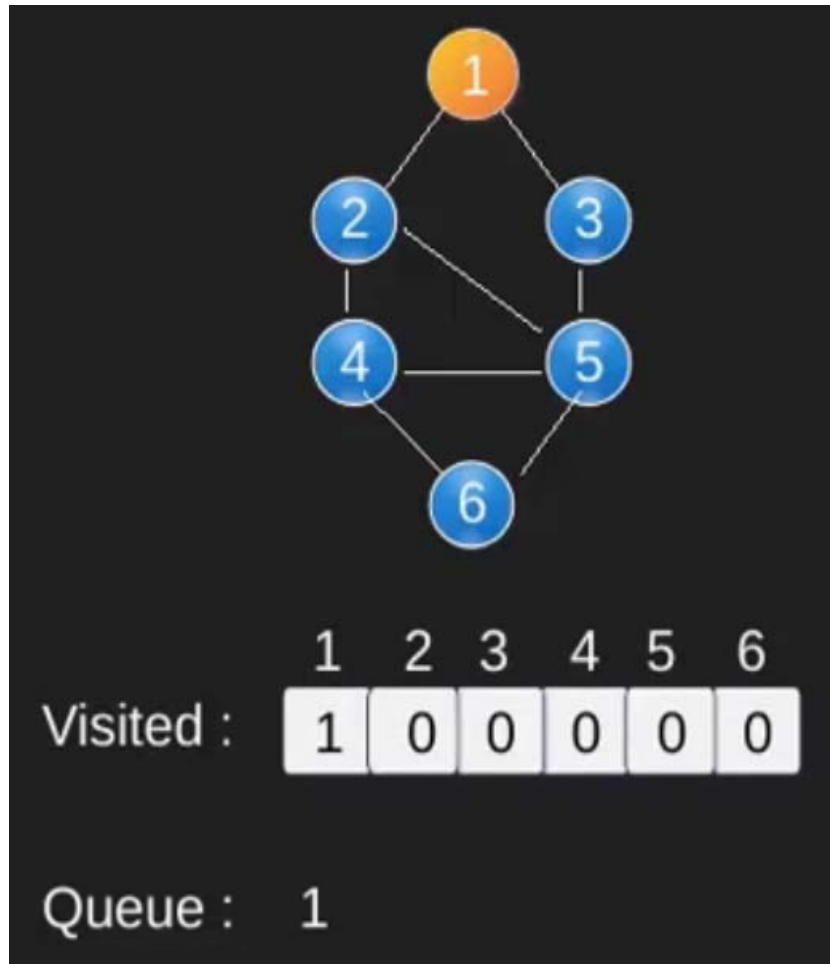
	0	1	2	3
0	0	1	1	0
1	1	0	0	1
2	1	0	0	0
3	0	1	0	0

# Breadth First Search (BFS)

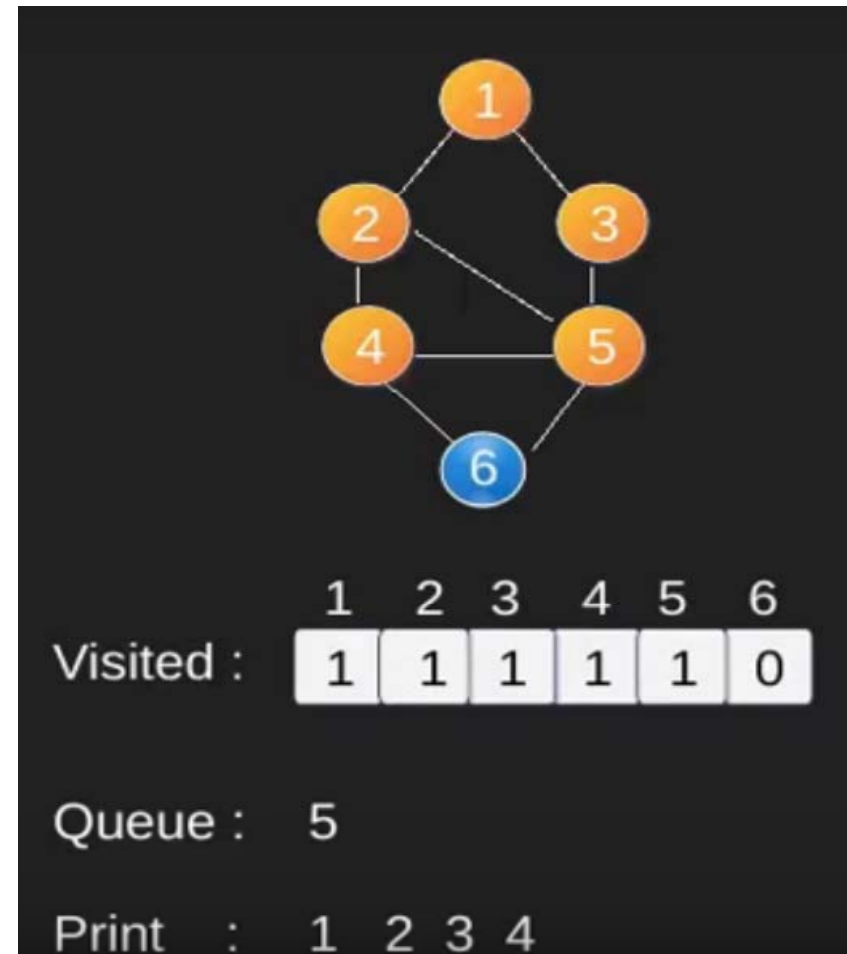
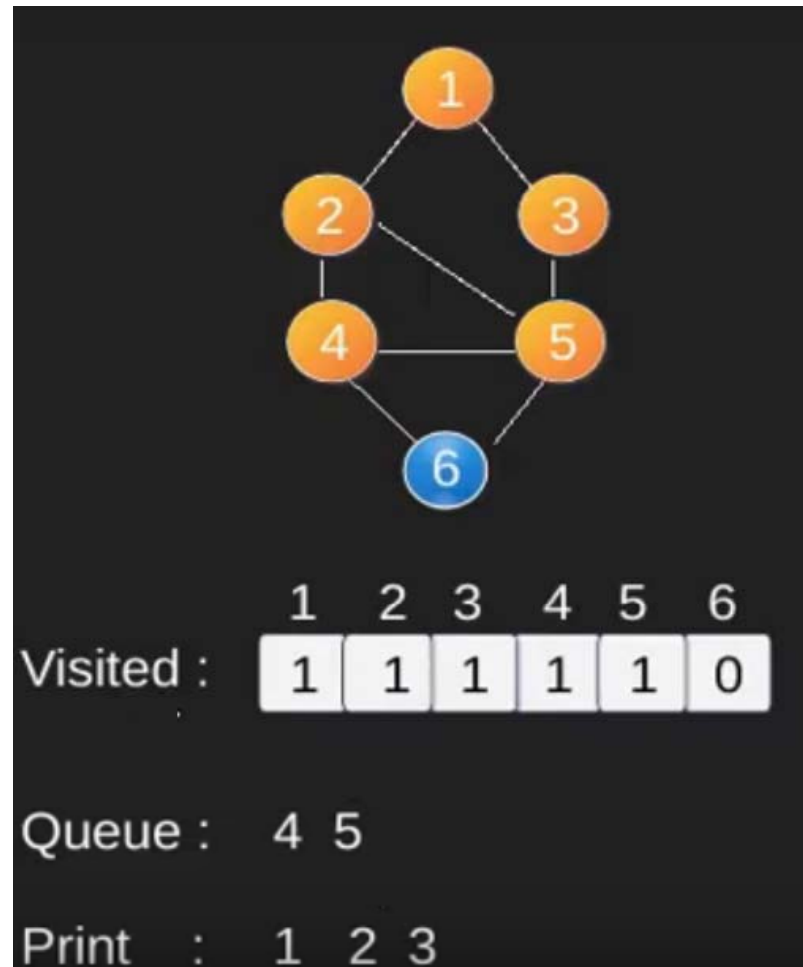


	A	B	C	D	E
A	0	1	1	0	0
B	1	0	0	1	0
C	1	0	0	0	1
D	0	1	0	0	0
E	0	0	1	0	0

## Breadth First Search (BFS) Example

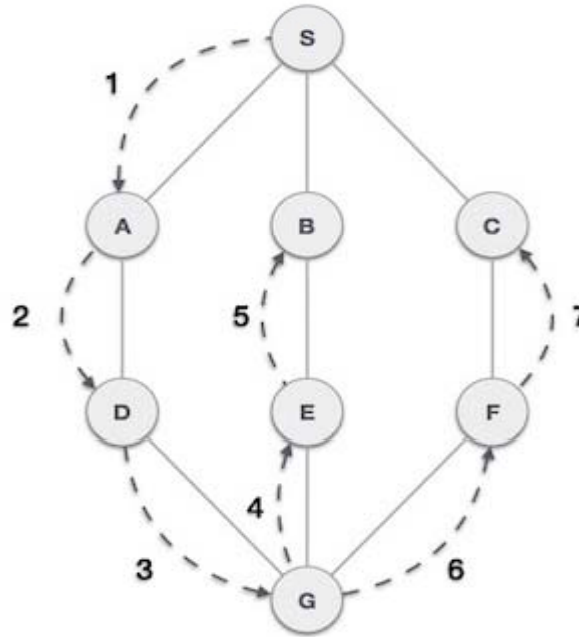


## Breadth First Search (BFS) Example





# Depth First Search (DFS)



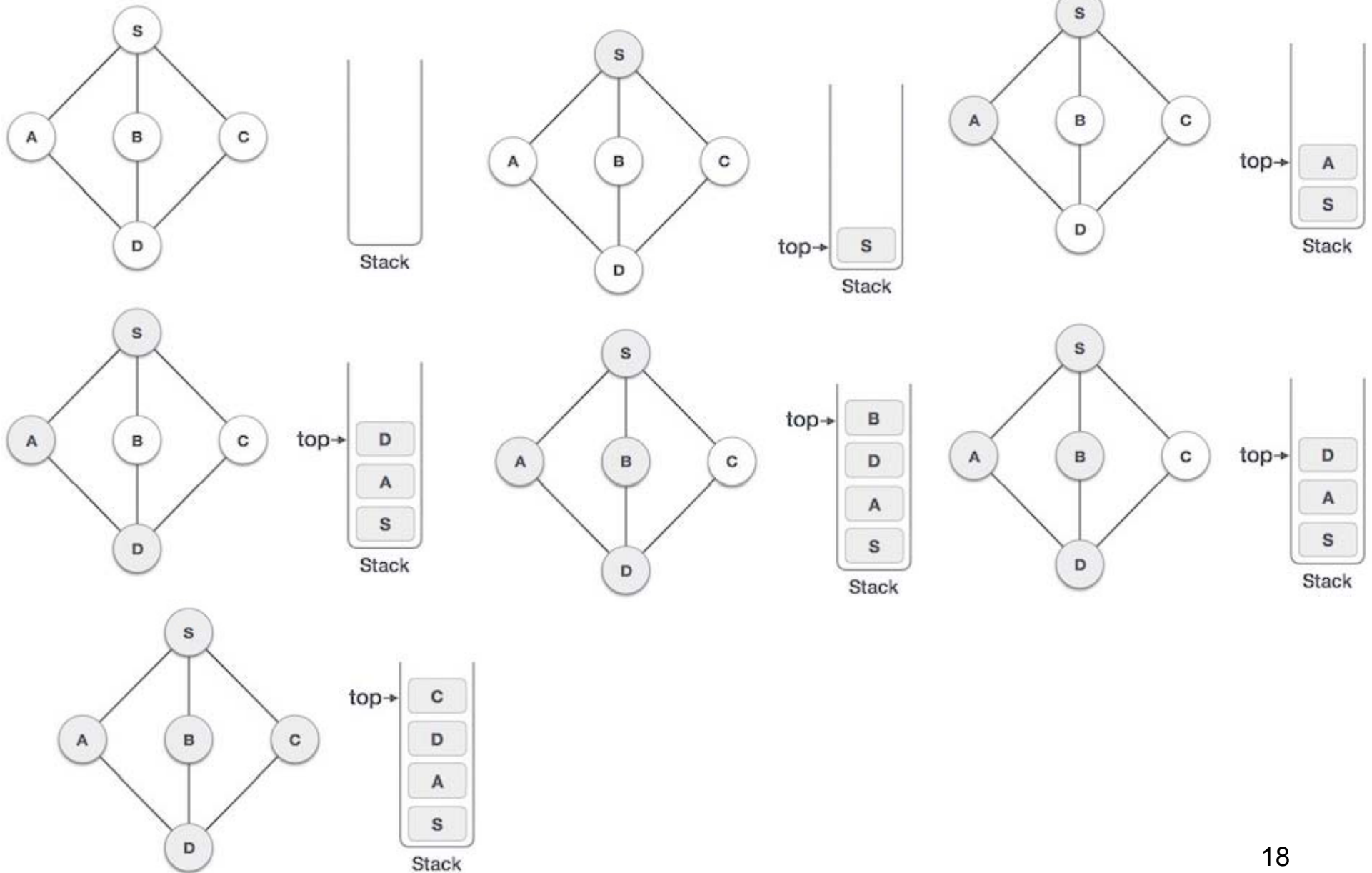
As in the example given above, DFS algorithm traverses from S to A to D to G to E to B first, then to F and lastly to C. It employs the following rules.

**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.

# Depth First Search



## 2 Depth First Search Step Two

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.

## 3 Depth First Search Step Three

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.

## 4 Depth First Search Step Four

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.

## 5 Depth First Search Step Five

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.

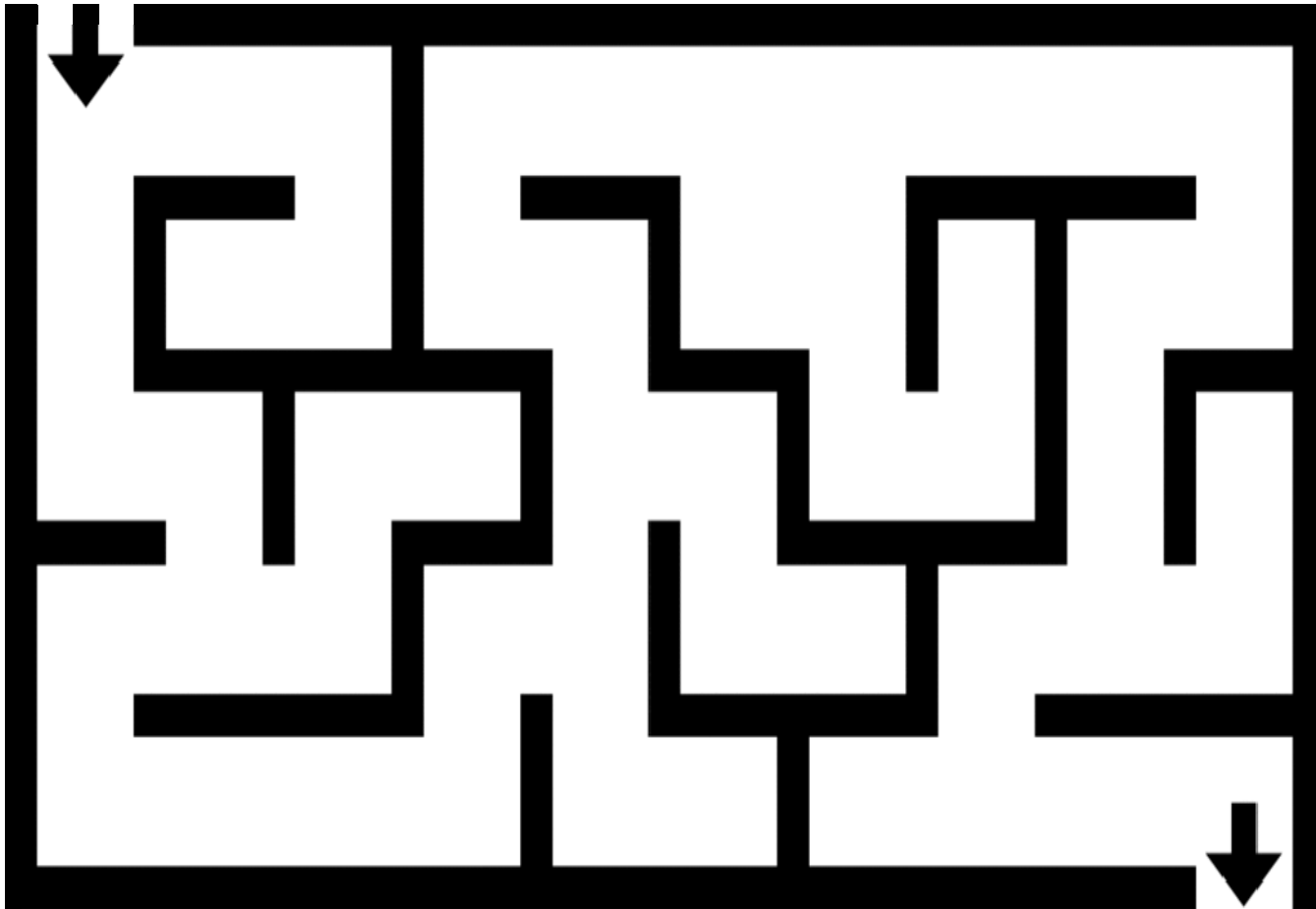
## 6 Depth First Search Step Six

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.

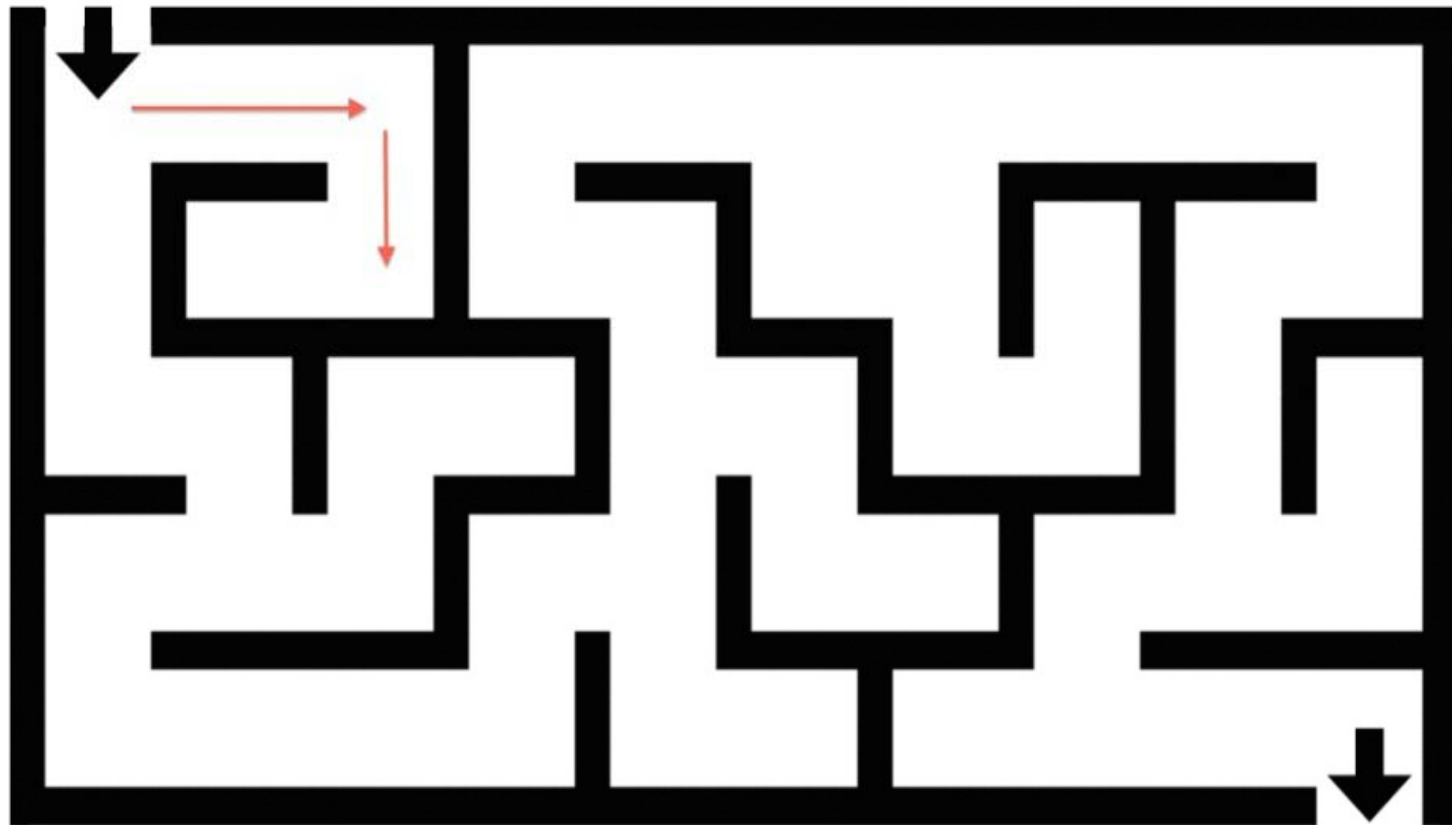
## 7 Depth First Search Step Seven

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

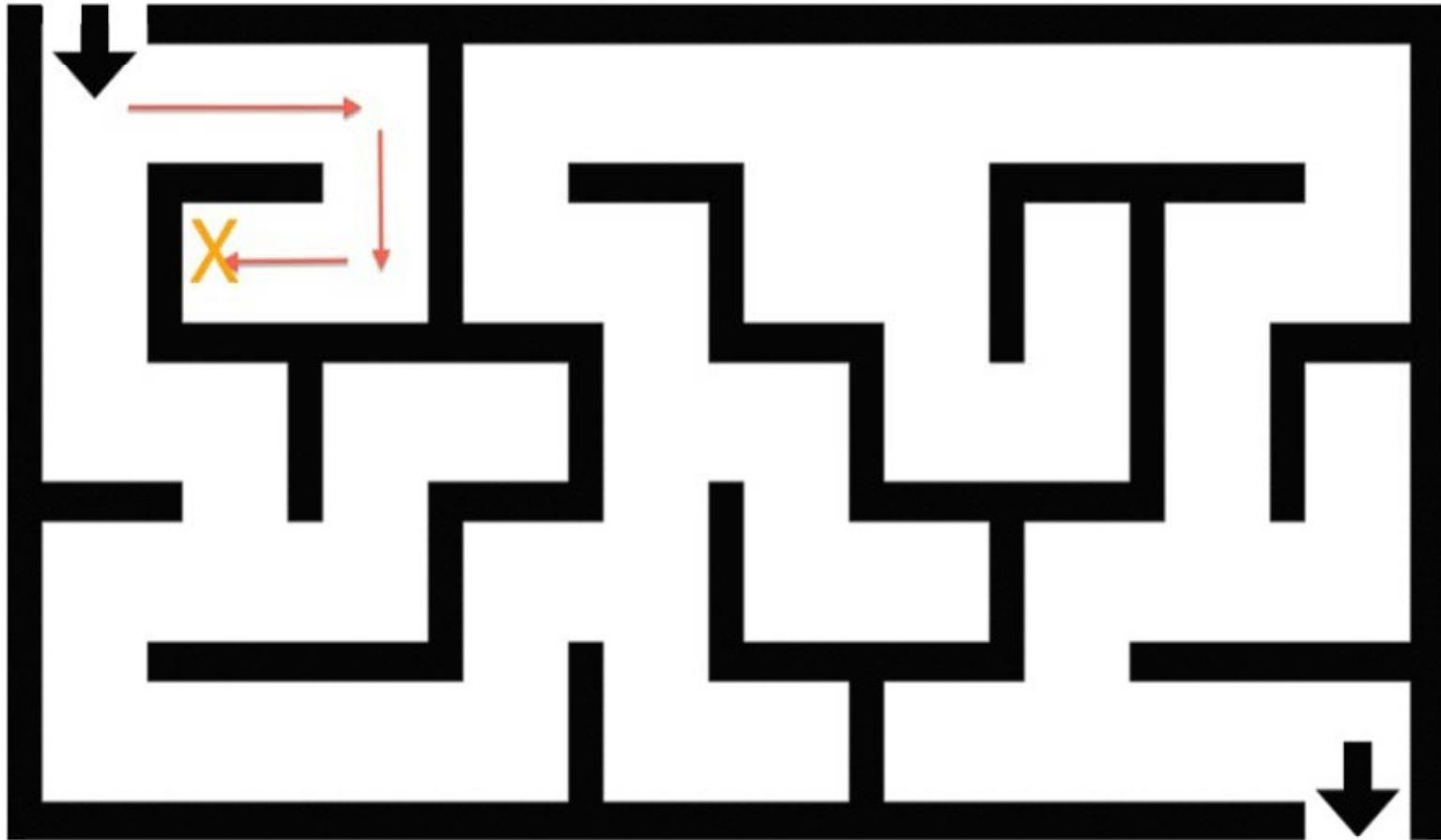
**How do you solve a maze?**



**How do you solve a maze?**



How do you solve a maze?



# Depth First Search Implementation

```
#include<stdio.h>
int a[20][20],visited[20],n;
int main()
{
    int v,i,j;

    printf("\n Enter the number of vertices:");
    scanf("%d",&n);

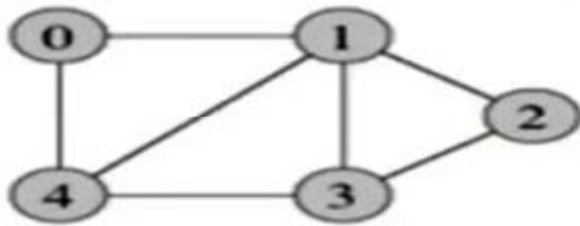
    for (i=0;i<n;i++)
    {
        visited[i]=0;
    }
    printf("\n Enter graph data in matrix form:\n");
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            scanf("%d",&a[i][j]);

    printf("\n Enter the starting vertex:");
    scanf("%d",&v);
    printf("\n DFS traversal is:\n");
    visited[v]=1; // mark the starting vertex as visited
    printf("%d  ",v);
    dfs(v);
}
```

Visited				
0	1	2	3	4
0 <sup>1</sup>	0	0	0	0

n=5  
v=0  
Print->

# Depth First Search Implementation



0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

One of the DFS traversal of the graph is  
0 1 2 3 4

Visited				
0	1	2	3	4
1	0	0	0	0

v=0, i=0 false  
v=0, i=1 print 1

v=2, i=0 false  
v=2, i=1 false  
v=2, i=2 false  
v=2, i=3 print 3

v=1, i=0 false  
v=1, i=1 false  
v=1, i=2 print 2

v=3, i=0 false  
v=3, i=1 false  
v=3, i=2 false  
v=3, i=3 false  
v=3, i=4 print 4

```

void dfs(int v)
{
    int i;
    for (i=0;i<n;i++)          // check all the vertices in the graph
    {
        if(a[v][i] != 0 && visited[i] == 0) // adjacent to v and not visited
        {
            visited[i]=1;          // mark the vertex visited
            printf("%d ",i);
            dfs(i);
        }
    }
}
  
```



# Differences between BFS and DFS.

Sr. No.	Key	BFS	DFS
1	Definition	BFS, stands for Breadth First Search.	DFS, stands for Depth First Search.
2	Data structure	BFS uses Queue to find the shortest path.	DFS uses Stack to find the shortest path.
3	Source	BFS is better when target is closer to Source.	DFS is better when target is far from source.
4	Suitability for decision tree	As BFS considers all neighbour so it is not suitable for decision tree used in puzzle games.	DFS is more suitable for decision tree. As with one decision, we need to traverse further to augment the decision. If we reach the conclusion, we won.
5	Speed	BFS is slower than DFS.	DFS is faster than BFS.
6	Time Complexity	Time Complexity of BFS = $O(V+E)$ where V is vertices and E is edges.	Time Complexity of DFS is also $O(V+E)$ where V is vertices and E is edges.

# Thank You

