Graph Traversal Methods:

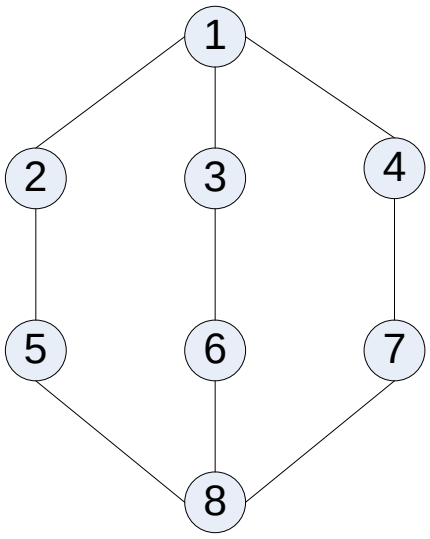
1) BFS : Breadth First Search

- Breadth First search (BFS) is an algorithm for traversing a graph.
- BFS uses queue for traversal.
- It starts from the some arbitrary node of a graph and explores all of the nodes level by level.

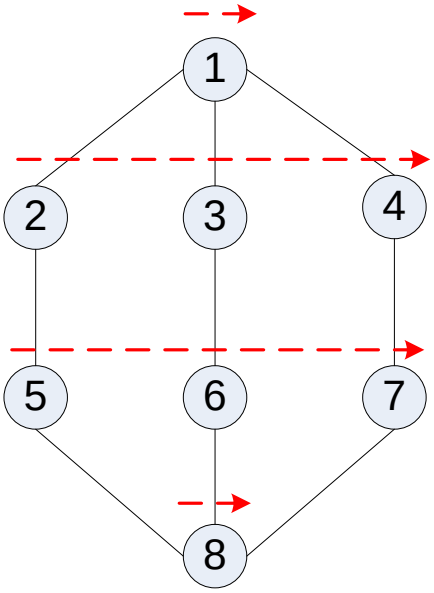
Algorithm : BFS(s)

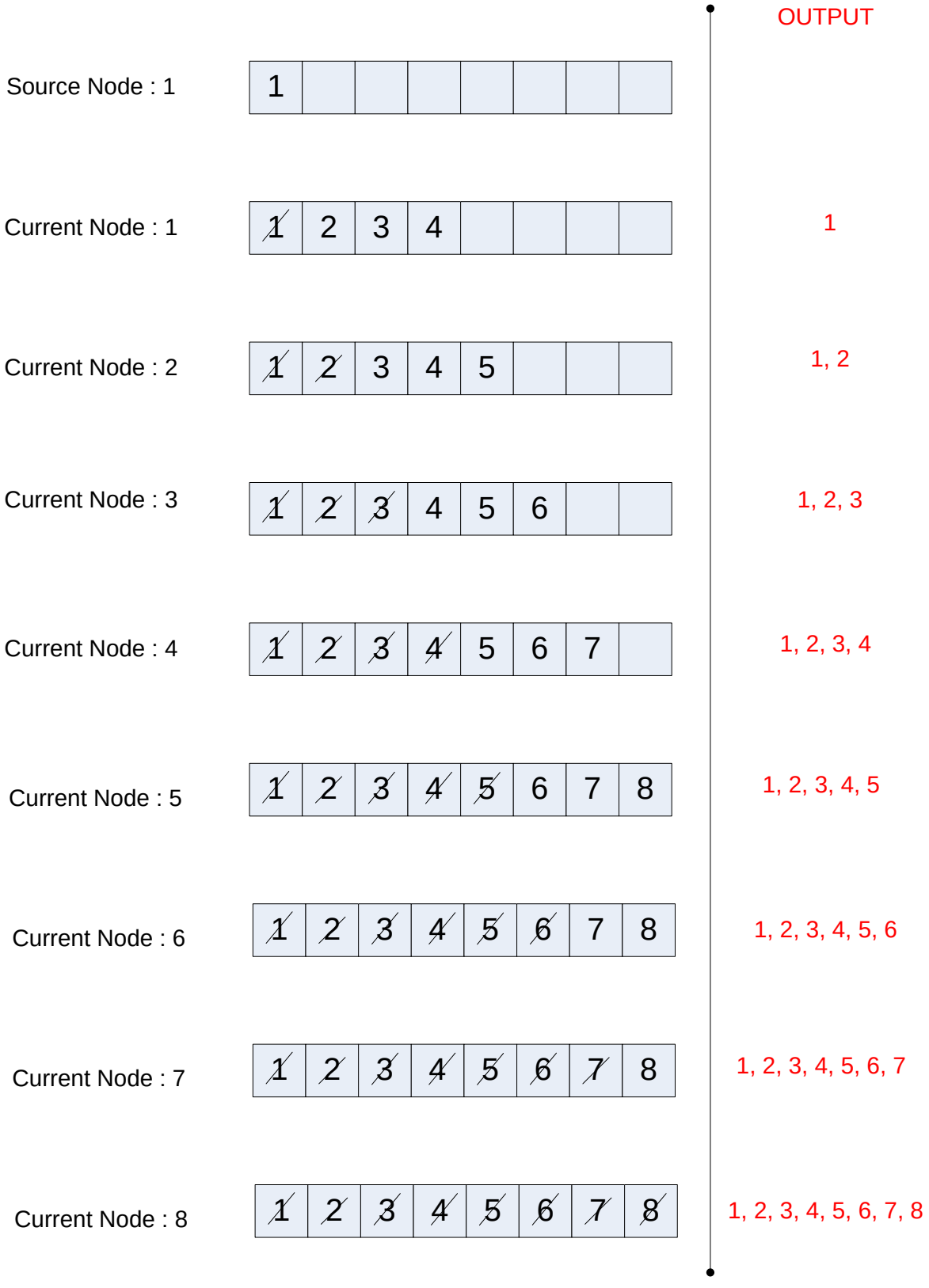
```
//  s – source node
//  n – no. of nodes in the graph
//  a[n][n] – adjacency matrix for the graph
//  visit[n] – an array for the visited nodes in the graph

Step 1:   enqueue(s)
Step 2:   visit[s]=1
Step 3:   p = dequeue()
Step 4:   while (p!=-1) repeat step 5 to 7
Step 5:   Print(p)
Step 6:   for(i=1 to n) do
          {       if( (a[p][i]=1) && (visit[i]=0) ) then
                  {
                          enqueue(i)
                          visit[i]=1
                  }
          } end for
Step 7: p = dequeue()
Step 8: Exit
```

Example: Traverse the following graph using BFS.



Solution :




OUTPUT
Source Node : 1
1
Current Node : 1
1 2 3 4
1
Current Node : 2
1 2 3 4 5
1, 2
Current Node : 3
1 2 3 4 5 6
1, 2, 3
Current Node : 4
1 2 3 4 5 6 7
1, 2, 3, 4
Current Node : 5
1 2 3 4 5 6 7 8
1, 2, 3, 4, 5
Current Node : 6
1 2 3 4 5 6 7 8
1, 2, 3, 4, 5, 6
Current Node : 7
1 2 3 4 5 6 7 8
1, 2, 3, 4, 5, 6, 7
Current Node : 8
1 2 3 4 5 6 7 8
1, 2, 3, 4, 5, 6, 7, 8

Output : 1, 2, 3, 4, 5, 6, 7, 8

Applications of BFS:

1) In Prim's algorithm to find Minimal Spanning Tree.
2) To find Single source Shortest Path (Dijkstra's Algorithm).
3) To find All Pairs Shortest Path (Floyd Warshall Algo).
4) Cycle detection in undirected graph.
5) To find all neighbor nodes in Peer to Peer Networks.
6) To find all neighboring locations in GPS Navigation systems.
7) To broadcast a packet in a Network.

2) DFS : Depth First Search

- Depth First search (BFS) is an algorithm for traversing a graph.
- DFS uses Stack for traversal.
- It starts from the some arbitrary node of a graph and explores a branch as deep as possible before backtracking and exploring another branch.

Algorithm : DFS(s)

// s – source node

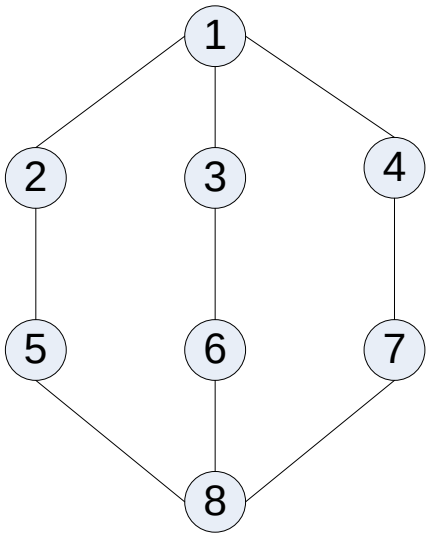
// n – no. of nodes in the graph

//a[n][n] – adjacency matrix for the graph

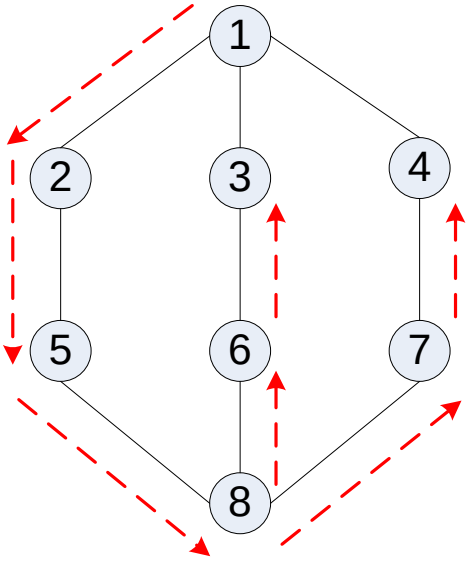
//visit[n] – an array for the visited nodes in the graph

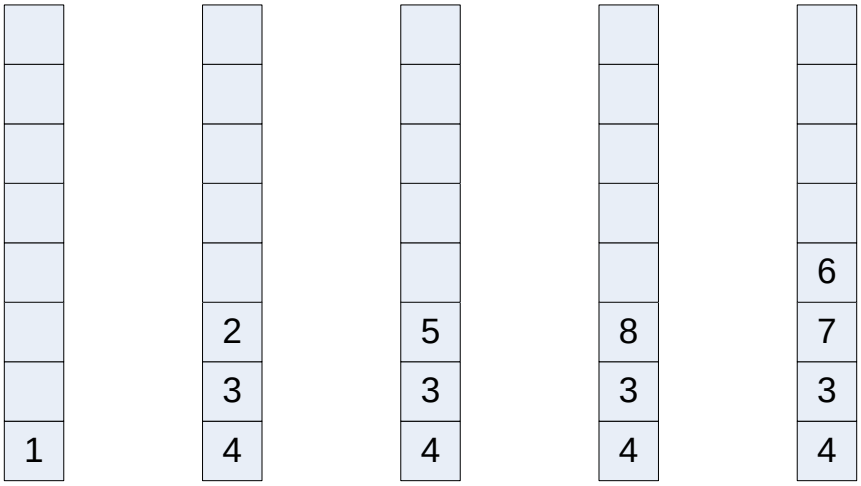
```
Step 1: push(s)
Step 2: k=pop()
Step 3: while(k!=-1) repeat step 4 to 6
Step 4: if(visit[k]=0) then
        {       Print(k)
                visit[k]=1
        }
 Step 5: for(i=n down to 1) do
        {
                if(a[k][i]=1 && visit[i]=0) then
                {
                        push(i)
                }
        }
Step 6: k=pop()
Step 7: Exit
```

Example : Traverse the following graph using DFS.



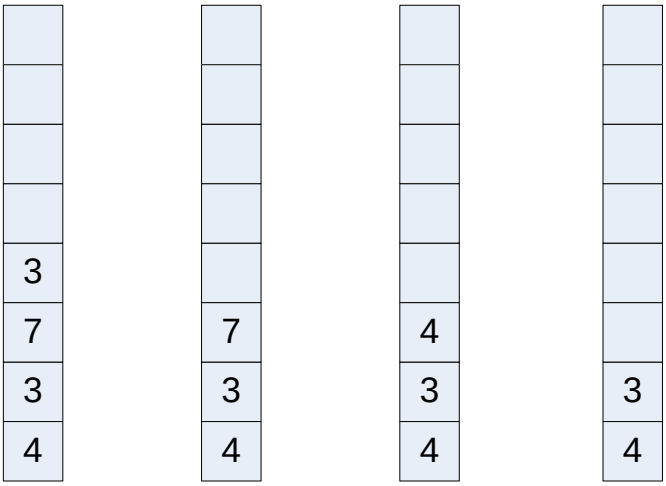
Solution :





Source Node : 1 | Current Node : 1 | Current Node : 2 | Current Node : 5 | Current Node : 8

OUTPUT : 1 | 1,2 | 1,2,5 | 1,2,5,8



Current Node : 6 | Current Node : 3 | Current Node : 7 | Current Node : 4

OUTPUT : 1,2,5,8,6 | 1,2,5,8,6,3 | 1,2,5,8,6,3,7 | 1,2,5,8,6,3,7,4

Output : 1, 2, 5, 8, 6, 3, 7, 4

Applications of DFS:

1) Cycle detection in directed and undirected graph.
2) To find Articulation Point in a graph.
3) To find strongly connected components in a graph.
4) For Topological sorting.
5) For Game playing.