Data Structures & Algorithms

Graph

Graph Traversal/Search

The goal of a graph traversal, generally, is to systematically explore all vertices reachable from a given start vertex in some particular order.

There are mainly two approach for graph traversal.

Depth-First Traversal/Depth-first Search (DFS).
Breadth-First Traversal/Breadth-first Search (BFS).

Depth First Search (DFS)

Given a connected, undirected and unweighted graph G = (V, E) and a designated source vertex s.

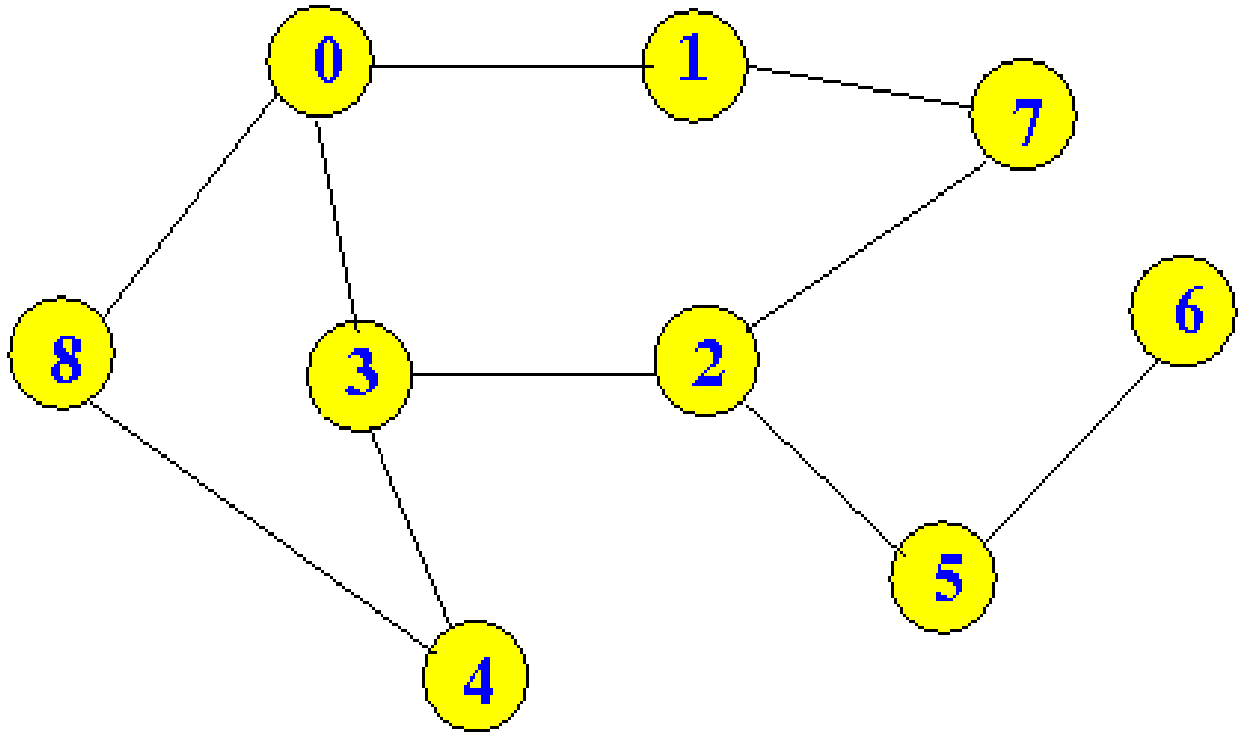
Idea: Starts at the source node (selecting any arbitrary node as the source node in a graph) and explores as far as possible along each branch before backtracking.

Go forward (in depth) while there is such possibility, otherwise to backtrack.
Go “as deep as possible“, keep going, when you can’t, backtrack to the one level back, then see if you can go anywhere else, if you can, go there and keep going, if not, backtrack to the one more level back, … and so on. When you come back to the source vertex, DFS ends.

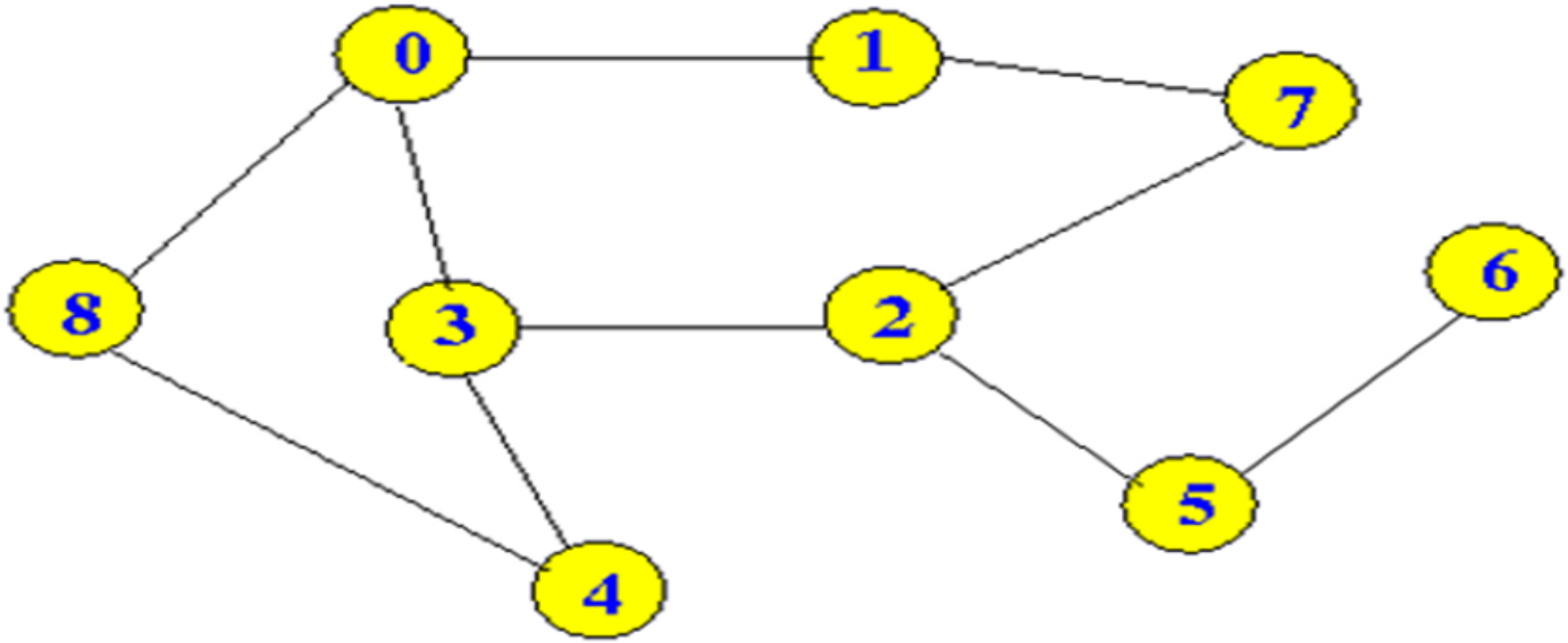
Stack is used in DFS traversal.

Depth First Search (DFS)

Consider following undirected graph. Start from node 0. Compute the DFS forest. Traverse the following graph using DFS and give DFS traversal sequence.



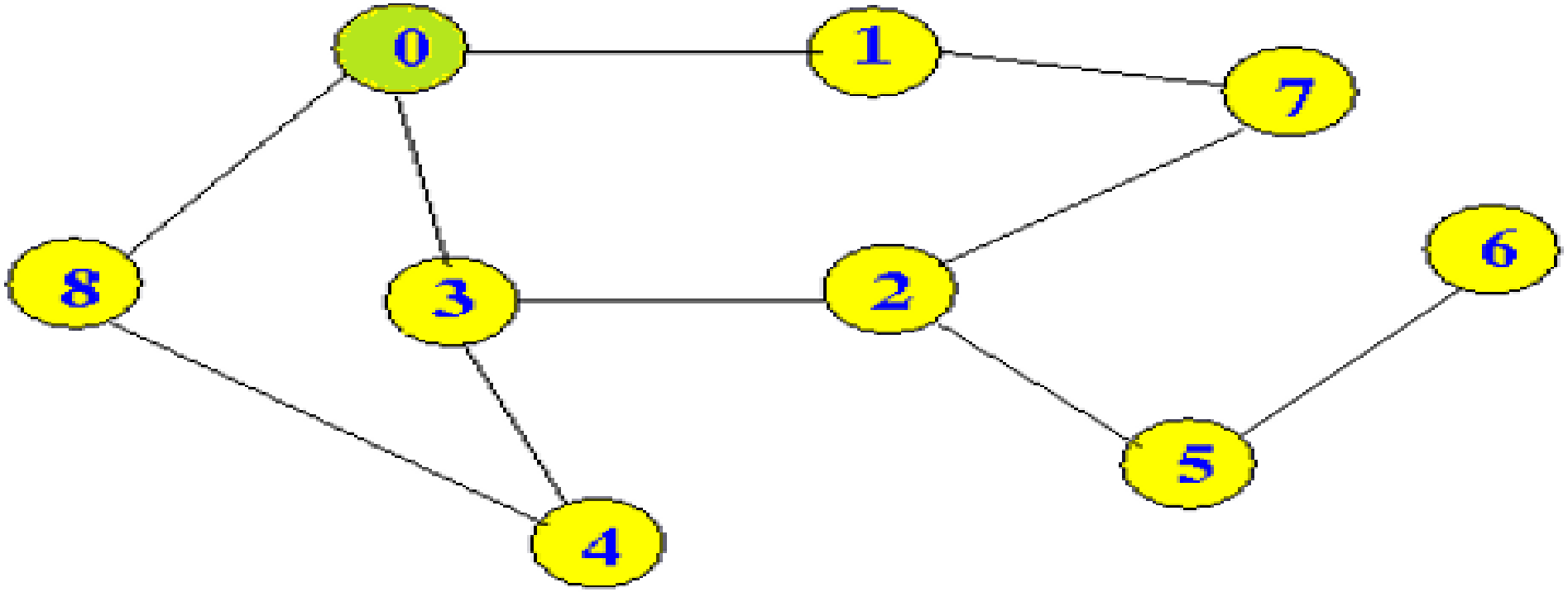
Depth First Search (DFS)



visited[]

0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0

Depth First Search (DFS)

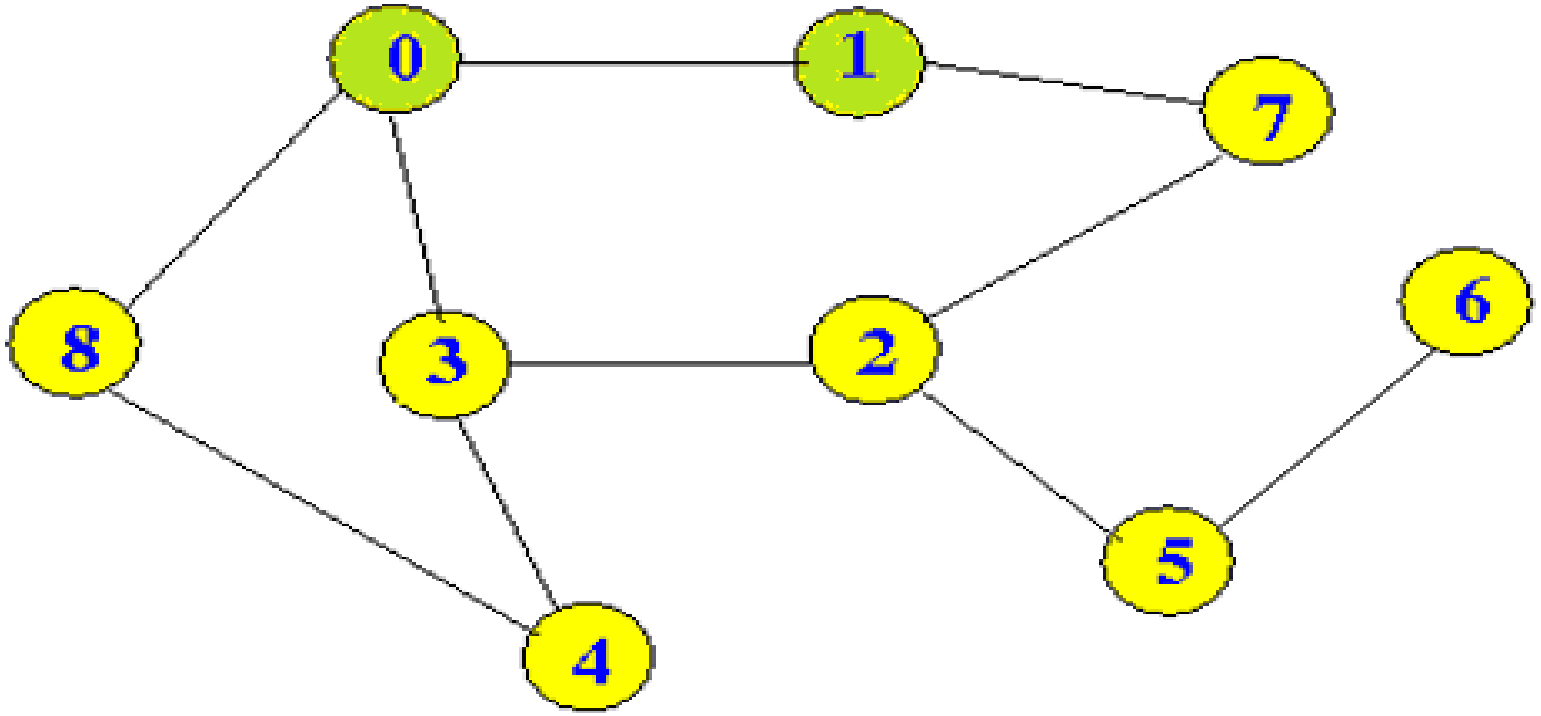


visited[]

0	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0

s = 0

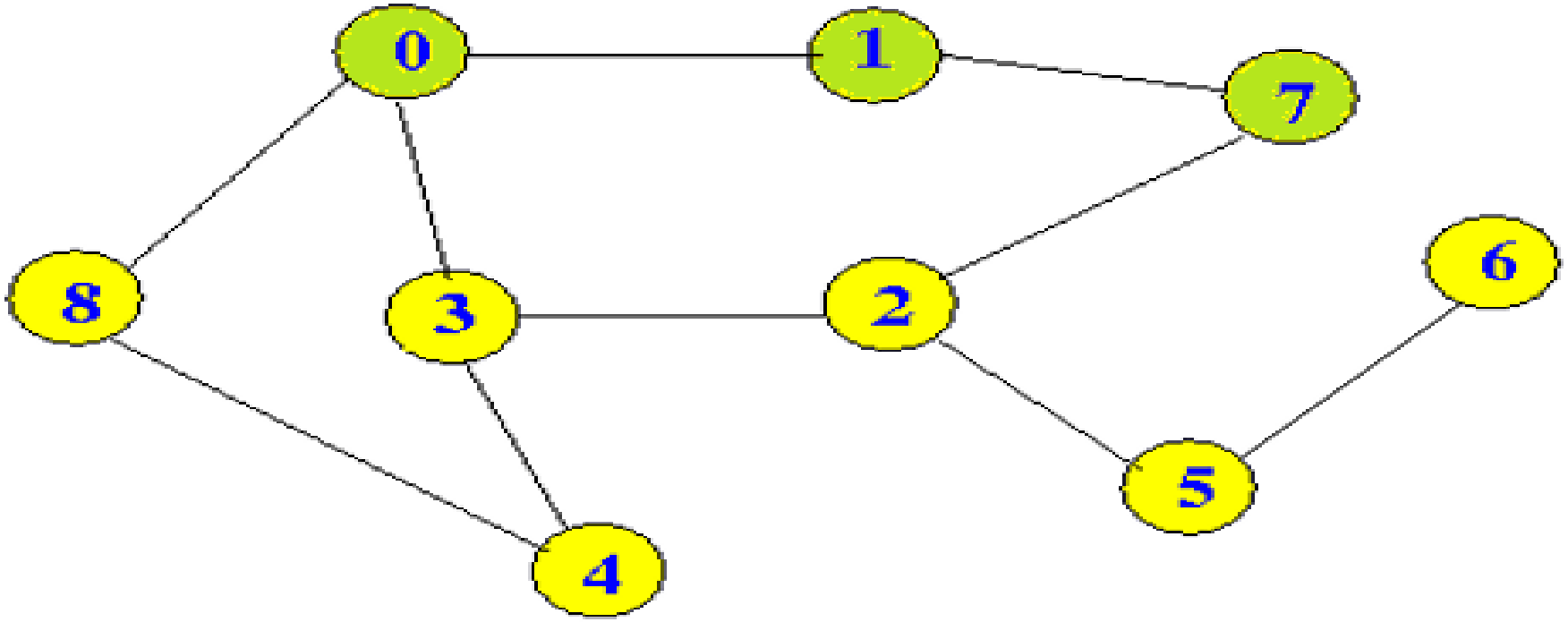
Depth First Search (DFS)



Now, neighbor(0) = {1, 3, 8}
So, visited[]

0	1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	0	0

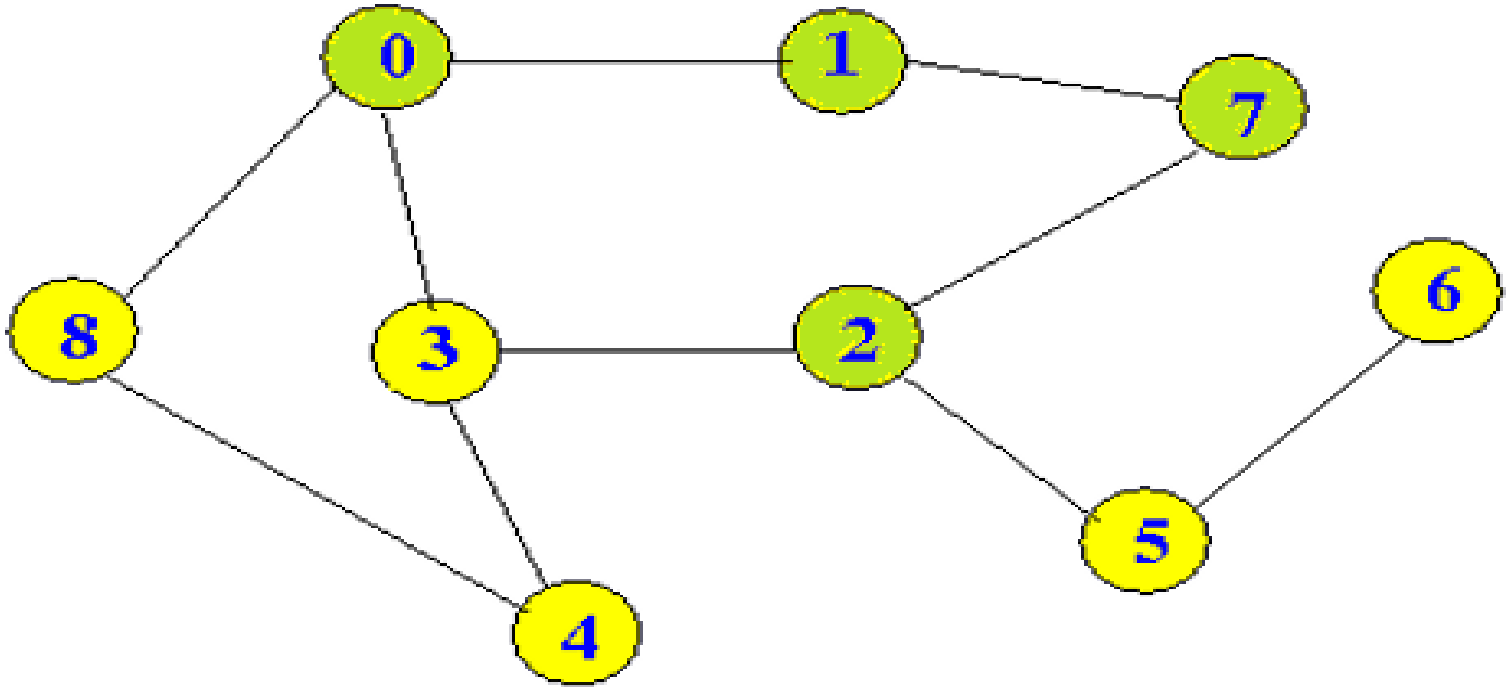
Depth First Search (DFS)



Now, neighbor(1) = {0, 7}
So, visited[]

0	1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	1	0

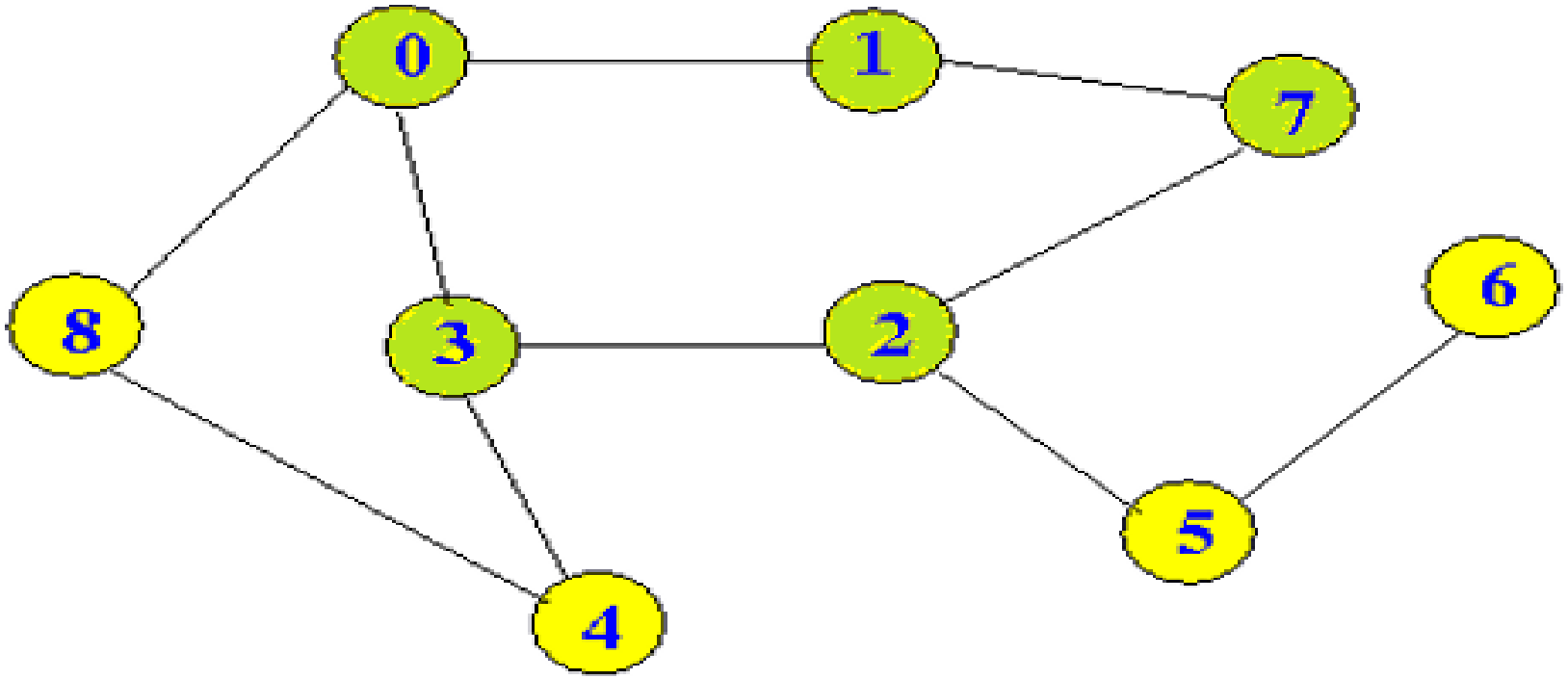
Depth First Search (DFS)



Now, neighbor(7) = {1, 2}
So, visited[]

0	1	2	3	4	5	6	7	8
1	1	1	0	0	0	0	1	0

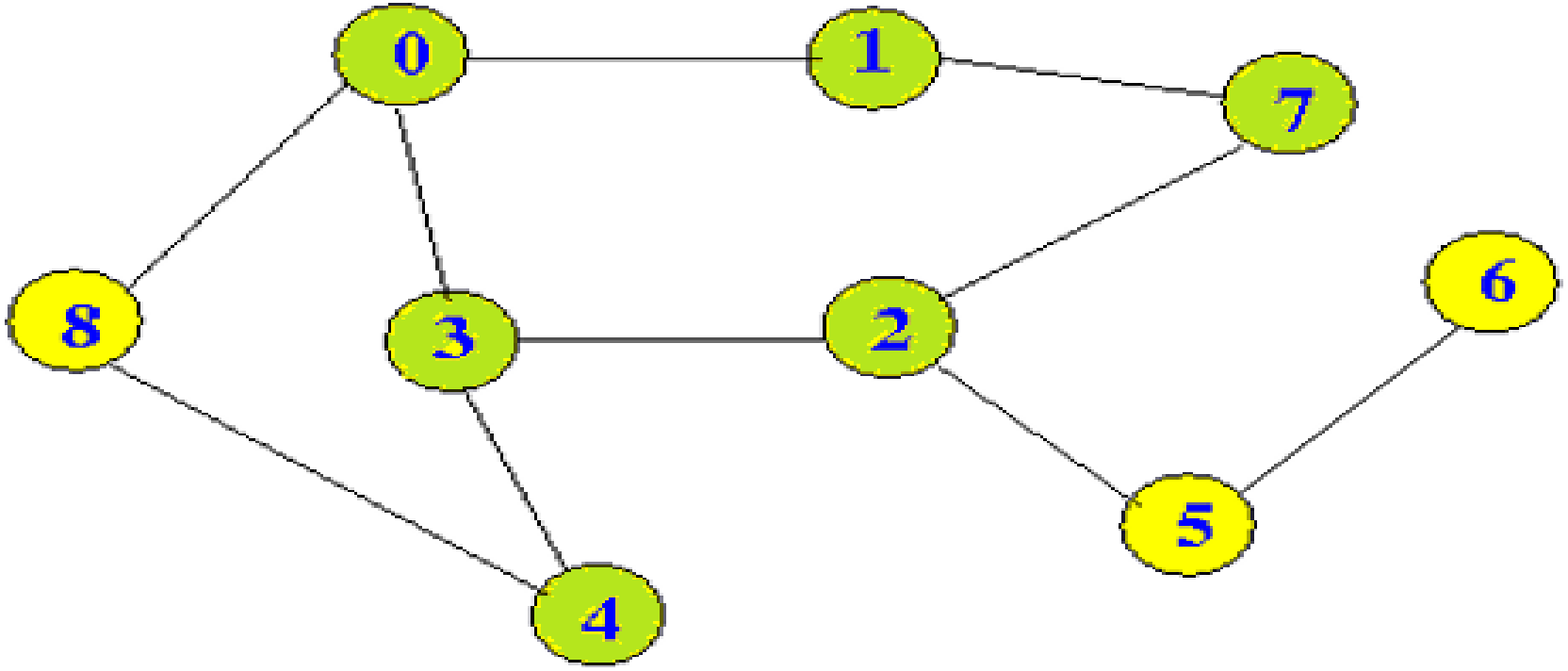
Depth First Search (DFS)



Now, neighbor(2) = {3, 5, 7}
So, visited[]

0	1	2	3	4	5	6	7	8
1	1	1	1	0	0	0	1	0

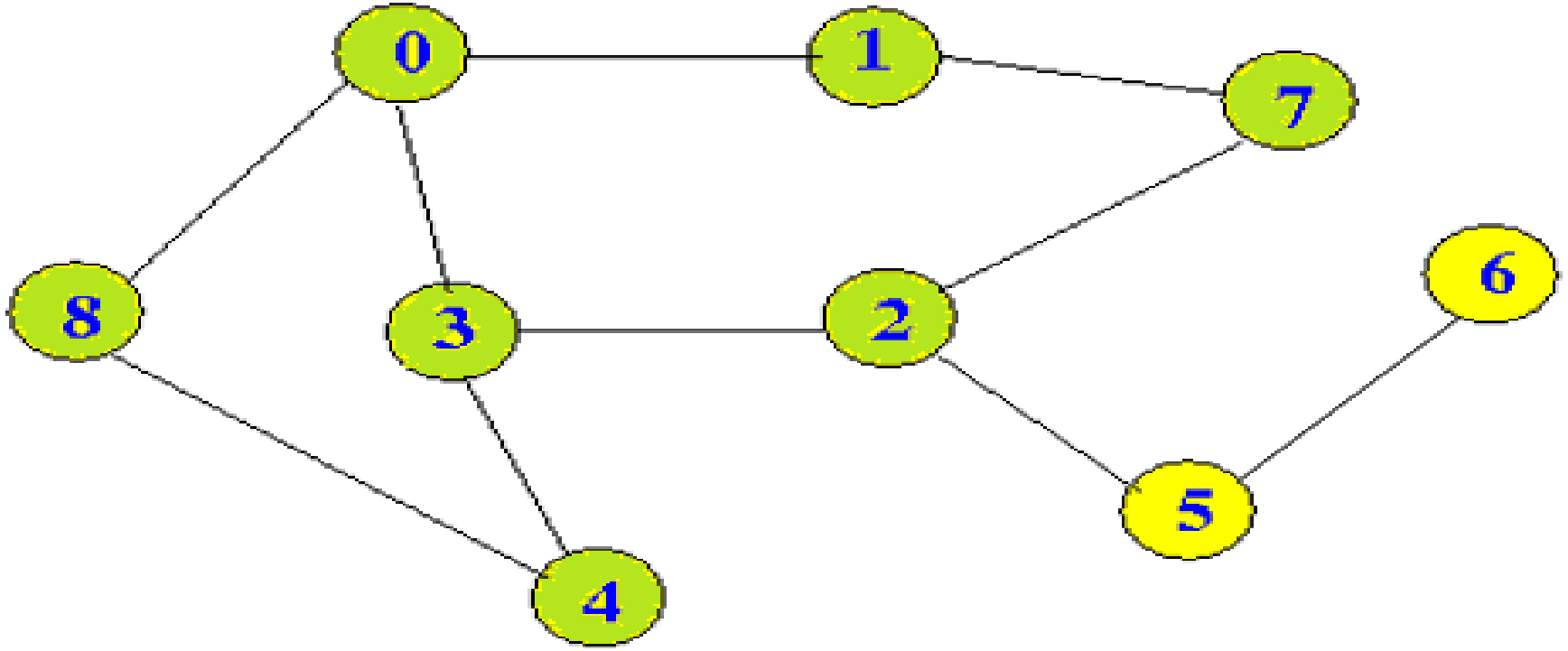
Depth First Search (DFS)



Now, neighbor(3) = {0, 2, 4}
So, visited[]

0	1	2	3	4	5	6	7	8
1	1	1	1	1	0	0	1	0

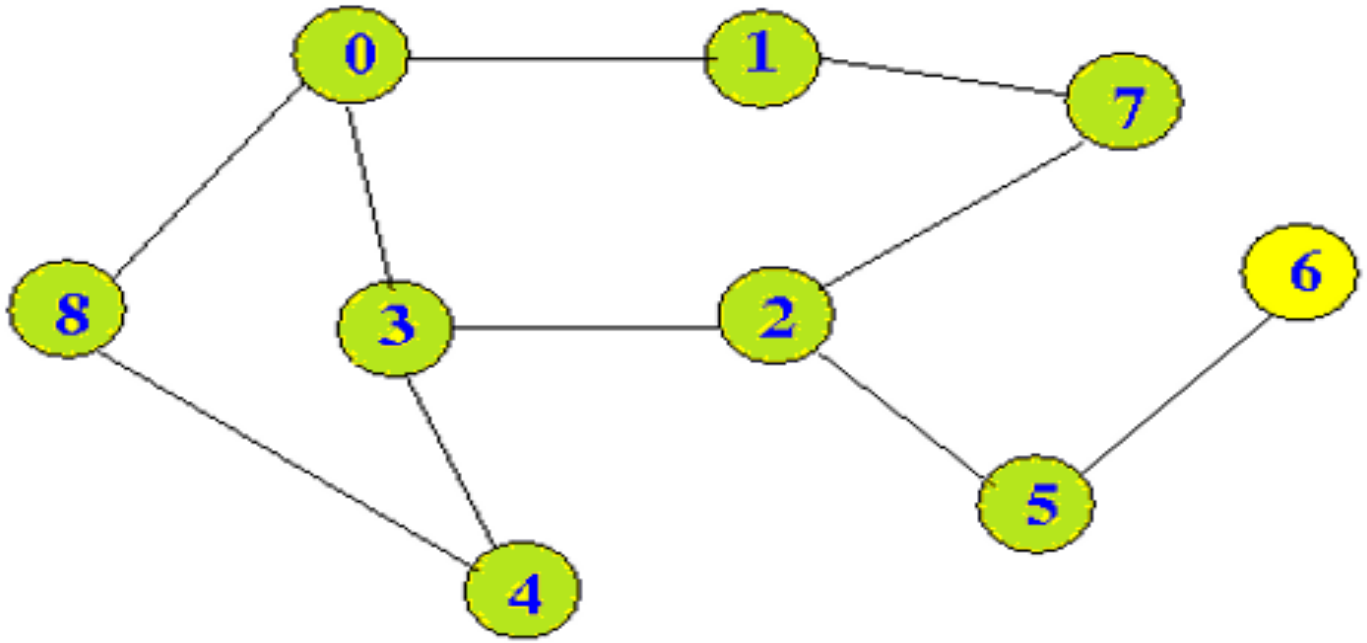
Depth First Search (DFS)



Now, neighbor(4) = {3, 8}
So, visited[]

0	1	2	3	4	5	6	7	8
1	1	1	1	1	0	0	1	1

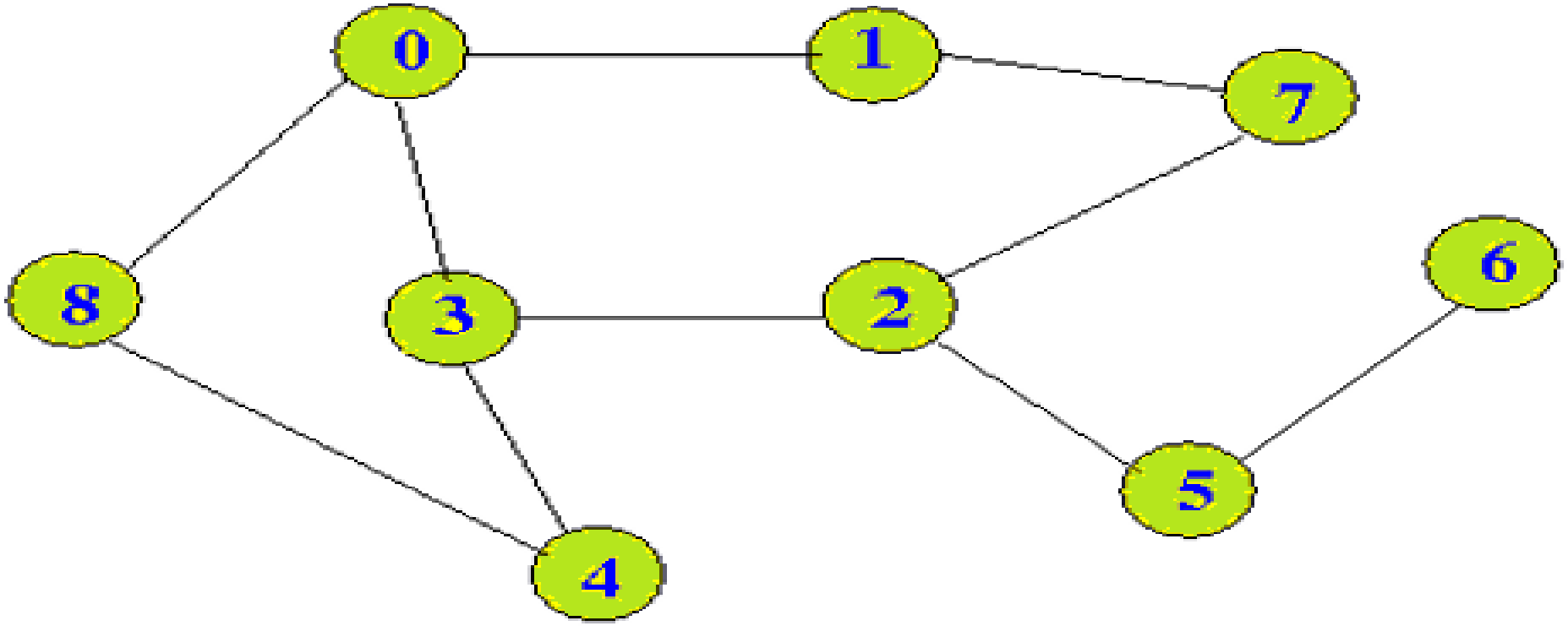
Depth First Search (DFS)



Now, neighbor(8) = {0, 4} already visited, so backtrack from node 8 to node 4, node 4 to node 3, node 3 to node 2.
Now, neighbor(2) = {7, 3, 5}
So, visited[]

0	1	2	3	4	5	6	7	8
1	1	1	1	1	1	0	1	1

Depth First Search (DFS)



Now, neighbor(5) = {2, 6}
So, visited[]

0	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1

Depth First Search (DFS)

Now, neighbor(6) = {5} already visited, so backtrack from node 6 to node 5, node 5 to node 2, node 2 to node 7, node 7 to node 1 and node 1 to node 0 and then the algorithm halt as all node of G is visited.

So, DFS traversal sequence is 0-1-7-2-3-4-8-5-6.

The DFS traversal sequence is not unique. Traversal sequence depend on the order that neighbor(s) are chosen.

Homework : Now, for the same undirected graph, start from node 8. Compute the DFS tree. Give DFS traversal sequence.

Depth First Search (DFS)

```
// G(V, E) is given undirected graph represented by adjacency list
// visited [i] is a binary array of size V, initialized by 0, indicates if vertex i is visited.
// neighbor(i) indicates {set of nodes to which vertex i is connected}
DFS-traversal(s)
{     // s is start node
   visited[s] = 1   // Changed the status of the node as visited
   print(s)
   for each node v in neighbor(s)
        if (visited [v] == 0)      // if node v is not visited yet
            DFS-traversal(v)   // Stack is maintained implicitly by recursive calls
}
```

Depth First Search (DFS)

Time complexity

It explores and marks every vertex once. Once a vertex is marked, it’s not explored again. It traverses each edge twice. Overall running time of the DFS procedure is O(V + E), linear in the size of the graph.

This assumes that the graph is represented as an adjacency list.