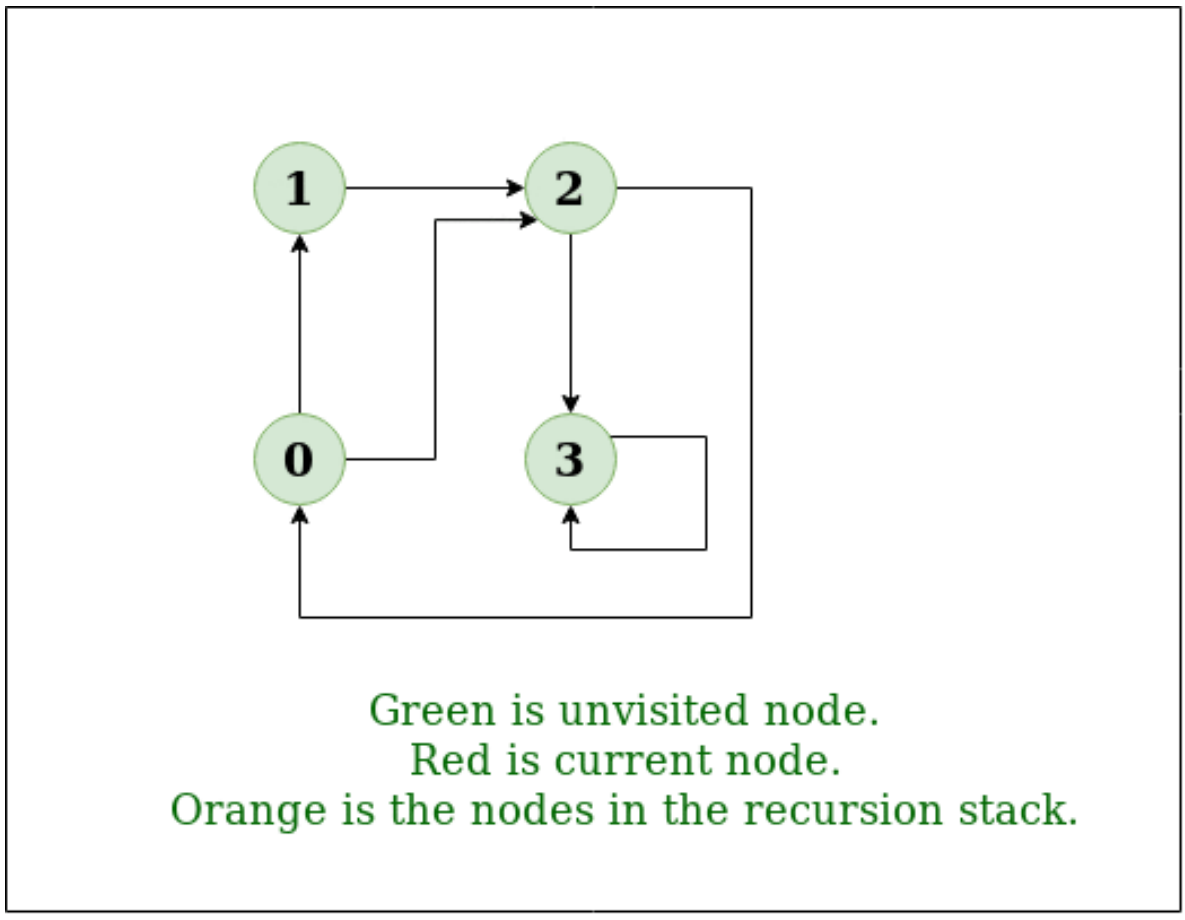
2.Depth First Search(DFS)

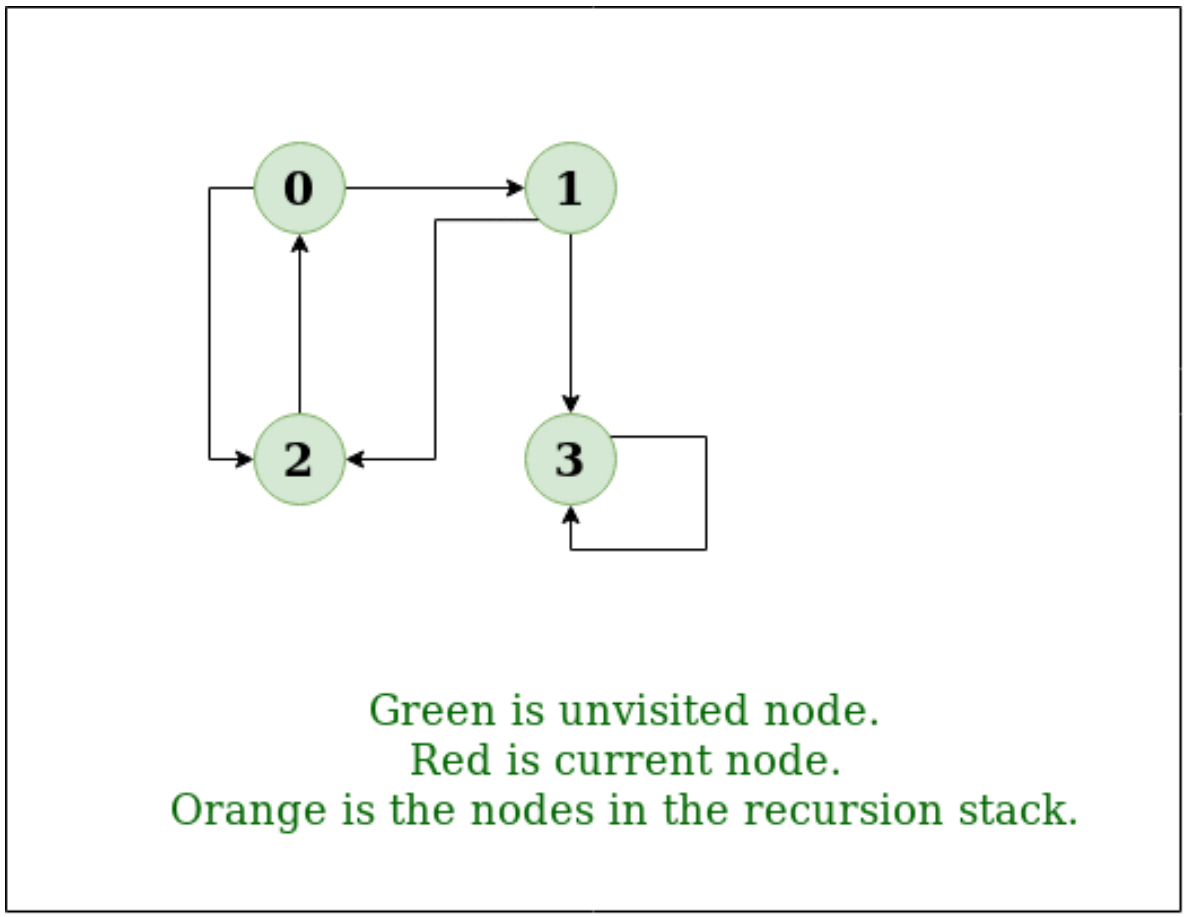
Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, a node may be visited twice. To avoid processing a node more than once, use a boolean visited array.

Example:

Input: *n = 4, e = 6*
0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3
Output: *DFS from vertex 1 : 1 2 0 3*
Explanation:
DFS Diagram:



Input: *n = 4, e = 6*
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3
Output: *DFS from vertex 2 : 2 0 1 3*
Explanation:
DFS Diagram:



- **Approach:** Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally print the nodes in the path.
- **Algorithm:**
  1. Create a recursive function that takes the index of node and a visited array.
  2. Mark the current node as visited and print the node.
  3. Traverse all the adjacent and unmarked nodes and call the recursive function with index of adjacent node.

==========
==========

```
# Python3 program to print DFS traversal
# from a given given graph
from collections import defaultdict
```

```
# This class represents a directed graph using
# adjacency list representation
class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # A function used by DFS
    def DFSUtil(self, v, visited):

        # Mark the current node as visited
        # and print it
        visited[v] = True
        print(v, end = ' ')

        # Recur for all the vertices
        # adjacent to this vertex
        for i in self.graph[v]:
            if visited[i] == False:
                self.DFSUtil(i, visited)

    # The function to do DFS traversal. It uses
    # recursive DFSUtil()
    def DFS(self, v):

        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph)+1)

        # Call the recursive helper function
        # to print DFS traversal
        self.DFSUtil(v, visited)

# Driver code

# Create a graph given
# in the above diagram
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
```

```
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is DFS from (starting from vertex 2)")
g.DFS(2)

# This code is contributed by Neelam Yadav
```