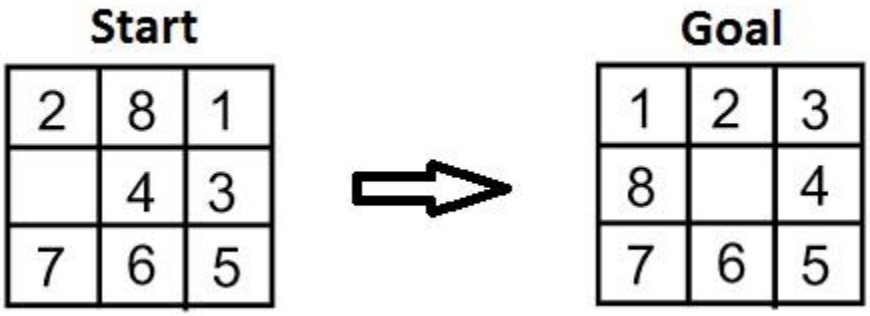
Task 1(A). Depth First Search solution of 8-puzzle problem

Solve following 8-puzzle problem using Depth First Search algorithm and print the visited node status of Depth First Traversal.



Input Format:

Index of nodes and edges of problem graph. For instance addEdge(source_Node, Destination_Node)

Sample Input:

addEdge(0, 1)

addEdge(0, 2)

addEdge(1, 2)

addEdge(2, 0)

addEdge(2, 3)

addEdge(3, 3)

Output Format:

Sequence of visited nodes of problem graph

Sample Output:

Following is Depth First Traversal (starting from vertex 2)

2 0 1 3

Sample Code:

```
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.add(v)
        print(v, end=' ')

        # Recur for all the vertices
        # adjacent to this vertex
        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)

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

        # Create a set to store visited vertices
        visited = set()

        # 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)
```