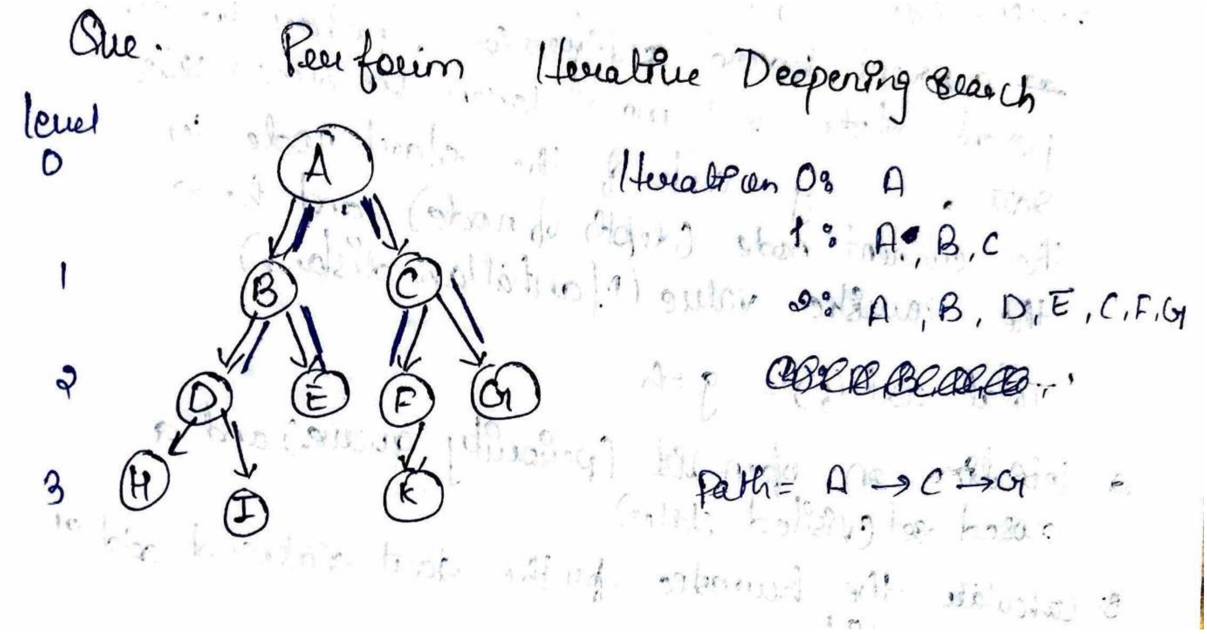
LAB PROGRAM 3

Implement Iterative deepening search algorithm.



Que. Perform Iterative Deepening Search

Iteration 0: A

1: A, B, C

2: A, B, D, E, C, F, G

Path = A → C → G

Iterative Deepening DFS:

```
graph = {
    'A' : ['B','C'],
    'B' : ['D','E'],
    'C' : ['G'],
    'D' : [],
    'E' : ['F'],
    'G' : [],
    'F' : []
}


def DFS(currNode, dest, graph, maxDepth, currList):
    print("Checking for ~~destination~~ destination", currNode)
    currList.append(currNode)
    if currNode == dest:
        return True
    if maxDepth <= 0:
        return False
    for node in graph[currNode]:
        if DFS(node, dest, graph, maxDepth -1, curlis
            return True
    currList.pop()   # Back track if no path found at
                       this depth
    return False

def iterativeDDFS( currNode, dest, graph, maxDepth)
  for i in range(maxDepth):
    print(f"\n --- Iteration with depth level(i) -"
    currlist =[]
    if DFS(currNode, dest, graph, i, currList):
        print ("Yes, path exists")
        print (currList)
        return True
    print(f" Completed level {i}, no path found at this
        depth.\n")
print ("Path not available")
return False

iterativeDDFS('A', 'E', graph, 4)
```

```
def possible_moves(state):
    b = state.index(0)
    directions = []
    if b not in [0,1,2]: directions.append('u')
    if b not in [6,7,8]: directions.append('d')
    if b not in [0,3,6]: directions.append('l')
    if b not in [2,5,8]: directions.append('r')
    return [(gen(state, d, b), d) for d in directions]

def gen(state, direction, b):
    temp = state.copy()
    if direction=='u': temp[b], temp[b-3] = temp[b-3], temp[b]
    if direction=='d': temp[b], temp[b+3] = temp[b+3], temp[b]
    if direction=='l': temp[b], temp[b-1] = temp[b-1], temp[b]
    if direction=='r': temp[b], temp[b+1] = temp[b+1], temp[b]

def print_board(state):
    board = np.array(state).reshape(3,3)
    print(board)

# Intial and target configuration
src = [1,2,3,0,4,5,6,7,8]
target=[1,2,3,4,5,6,7,8,0]

# Run bfs to solve the puzzle
bfs(src, target)
```

Code:

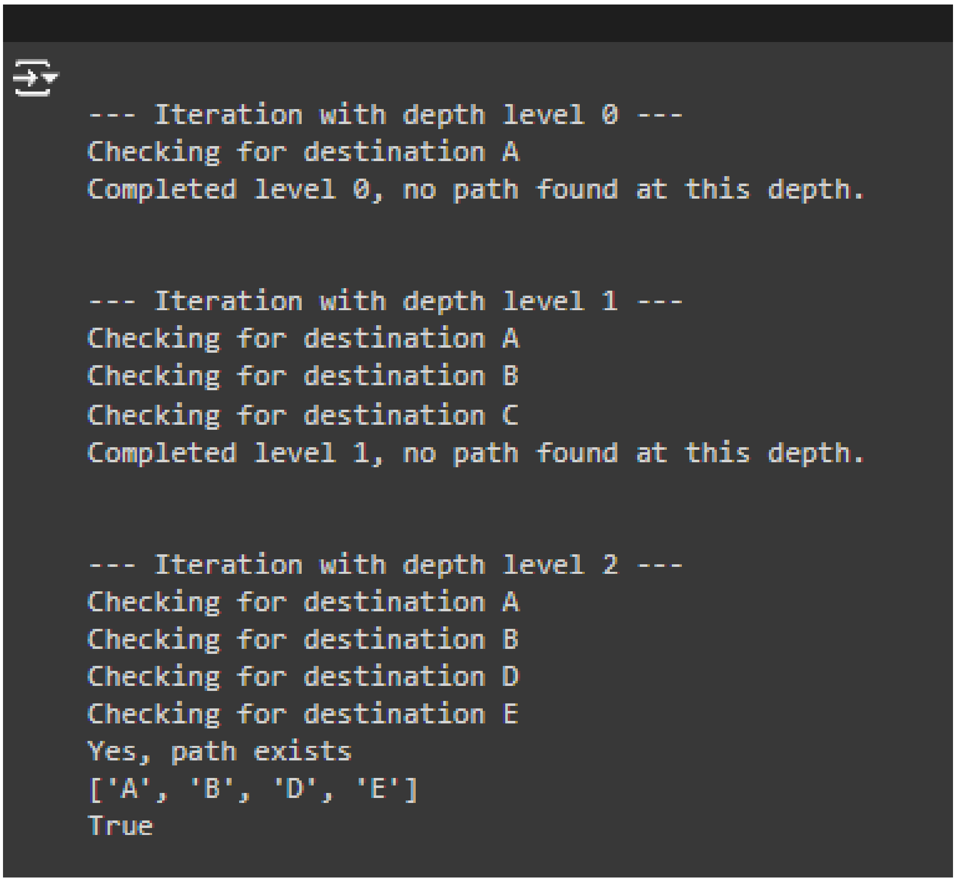
```
graph = {
    'A': ['B', 'C'],
    'B': ['D','E'],
    "C": ['G'],
    'D': [],
    'E': ['F'],
    'G': [],
    'F':[]
}

def DFS(currentNode, destination, graph, maxDepth, curList):
    print("Checking for destination", currentNode)
    curList.append(currentNode)
    if currentNode == destination:
        return True
    if maxDepth <= 0:
        return False
    for node in graph[currentNode]:
        if DFS(node, destination, graph, maxDepth - 1, curList):
            return True
    curList.pop()  # Backtrack if no path is found at this depth
    return False

def iterativeDDFS(currentNode, destination, graph, maxDepth):
    for i in range(maxDepth):
        print(f"\n--- Iteration with depth level {i} ---")
        curList = []
        if DFS(currentNode, destination, graph, i, curList):
            print("Yes, path exists")
            print(curList)
            return True
        print(f"Completed level {i}, no path found at this depth.\n")
    print("Path is not available")
    return False

# Calling the function
iterativeDDFS('A', 'E', graph, 4)
```

Output:



```
--- Iteration with depth level 0 ---
Checking for destination A
Completed level 0, no path found at this depth.


--- Iteration with depth level 1 ---
Checking for destination A
Checking for destination B
Checking for destination C
Completed level 1, no path found at this depth.


--- Iteration with depth level 2 ---
Checking for destination A
Checking for destination B
Checking for destination D
Checking for destination E
Yes, path exists
['A', 'B', 'D', 'E']
True
```