*Python program for BFS (RECURSIVE)

```
from collections import deque



class Graph:
        def __init__(self, edges, n):



        self.adjList = [[] for _ in range(n)]


                for (src, dest) in edges:
            self.adjList[src].append(dest)
            self.adjList[dest].append(src)



# Perform BFS recursively on the graph
def recursiveBFS(graph, q, discovered):


    if not q:
        return


    # dequeue front node and print it
    v = q.popleft()
    print(v, end=' ')


    # do for every edge (v, u)
    for u in graph.adjList[v]:
        if not discovered[u]:
            # mark it as discovered and enqueue it
            discovered[u] = True
            q.append(u)

    recursiveBFS(graph, q, discovered)


if __name__ == '__main__':


    edges = [
        (0, 1), (0, 2), (1, 2), (2, 0), (2, 3), (3, 3),


    ]

    # total number of nodes in the graph (labelled from 0 to 4)
    n = 4

        graph = Graph(edges, n)


    discovered = [False] * n

    # create a queue for doing BFS
    q = deque()

    # Perform BFS traversal from all undiscovered nodes to
        for i in range(n):
        if not discovered[i]:
```

```
                        discovered[i] = True

                        q.append(i)

            # start BFS traversal from vertex i
            recursiveBFS(graph, q, discovered)
```

*OUTPUT : 0 1 2 3

*Python program for DFS (RECURSIVE)

```
from collections import defaultdict


class Graph:


    def __init__(self):


        self.graph = defaultdict(list)
```

```
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.
    # recursive DFSUtil()
    def DFS(self, v):

        visited = set()

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

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(0)
```

OUTPUT : 0 1 2 3