BFS CODE:

```
from collections import deque


# A class to represent a graph object
class Graph:
    # Constructor
    def __init__(self, edges, n):
        self.adjList = [[] for _ in range(n)]

        # add edges to the undirected graph
        for (src, dest) in edges:
            self.adjList[src].append(dest)
            self.adjList[dest].append(src)


# Function to 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__':

    # List of graph edges as per the above diagram
    #edges = [
        # Notice that node 0 is unconnected
        #(1, 8), (1, 5), (1, 2), (8, 6), (8, 4), (8, 3),
        #(6, 10), (6, 7), (2, 9)
```

```
    #]
    edges = list(tuple(map(int,input().split())) for r in
range(int(input("Enter edges:"))))
    print(edges)

    # total number of nodes in the graph
    #n = 11
    n = int(input("Enter value of n:"))

    # build a graph from the given edges
    graph = Graph(edges, n)

    # to keep track of whether a vertex is discovered or
not
    discovered = [False] * n

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

    # Perform BFS traversal from all undiscovered nodes
    print("\nFollowing is Breadth First Traversal: ")
    for i in range(n):
        if not discovered[i]:
            # mark the source vertex as discovered
            discovered[i] = True

            # enqueue source vertex
            q.append(i)

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

OUTPUT:

```
Enter edges:9
1 8
1 5
1 2
8 6
8 4
8 3
6 10
6 7
2 9
[(1, 8), (1, 5), (1, 2), (8, 6), (8, 4), (8, 3), (6, 10), (6, 7), (2, 9)]
Enter value of n:11

Following is Breadth First Traversal:
0 1 8 5 2 6 4 3 9 10 7

...Program finished with exit code 0
Press ENTER to exit console.
```

DFS CODE:

```
class Graph:
	# Constructor
	def __init__(self, edges, n):
		self.adjList = [[] for _ in range(n)]

		# add edges to the undirected graph
		for (src, dest) in edges:
			self.adjList[src].append(dest)
			self.adjList[dest].append(src)


# Function to perform DFS recursively on the graph
def recursive_DFS(graph, v, discovered):

	discovered[v] = True			# mark the current node as discovered
	print(v, end=' ')			# print the current node

	# do for every edge (v, u)
	for u in graph.adjList[v]:
		if not discovered[u]:	# if `u` is not yet discovered
			recursive_DFS(graph, u, discovered)


if __name__ == '__main__':

	# List of graph edges as per the above diagram
	edges = list(tuple(map(int, input().split())) for r in range(int(input("Enter edges:"))))
	print(edges)
	#edges = [
		# Notice that node 0 is unconnected
		#(1, 2), (1, 3), (2, 4), (2, 5), (4, 6), (6, 7), (3, 5), (5, 6)
	#]

	# total number of nodes in the graph
```

```
#n = 8
n = int(input("Enter value of n:"))

graph = Graph(edges, n)

# to keep track of whether a vertex is discovered or not
discovered = [False] * n

# Perform DFS traversal from all undiscovered nodes
print("\nFollowing is Depth First Traversal: ")
for i in range(n):
        if not discovered[i]:
                recursive_DFS(graph, i, discovered)
```

OUTPUT:

```
Enter edges:8
1 2
1 3
2 4
2 5
4 6
6 7
3 5
5 6
[(1, 2), (1, 3), (2, 4), (2, 5), (4, 6), (6, 7), (3, 5), (5, 6)]
Enter value of n:8

Following is Depth First Traversal:
0 1 2 4 6 7 5 3

...Program finished with exit code 0
Press ENTER to exit console.
```