```python
def creategraph():
    n = int(input("Enter the number of vertices in the graph: "))
    adj_list = {}

    for i in range(n):
        print("Enter adjacent vertices of vertex", i)
        # Prompt the user to input adjacent vertices, split the input string by whitespace,
        # convert each element to an integer, and store them in a list.
        adj_vertices = list(map(int, input().split()))

        adj_list[i] = adj_vertices

    return adj_list

# Take user input for graph
adj_list = creategraph()
print(adj_list)


# ********************************************************************** #

# DFS (recursive)
"""
    adj_list =
    {
        0: [1, 3, 5],
        1: [0, 2],
        2: [1],
        3: [0, 4]
        4: [3],
        5: [0, 6],
        6: [5]
    }

"""

visited = []
def DFS(v):
    # Mark the current node as visited and print it
    visited.append(v)
    print(v, end=" ")

    # Recur for all the vertices adjacent to this vertex
    for neighbour in adj_list[v]:
        # Check if the neighbour vertex has not been visited yet
        if neighbour not in visited:
            # If not visited, recursively call DFS for the neighbour vertex
            DFS(neighbour)


# Call the DFS with starting node 0
DFS(0)

# ********************************************************************** #

# BFS (recursive)

# Initialize the visited list and mark the starting node as visited

'''

    adj_list =
    {
        0: [1, 3, 5],
        1: [0, 2],
        2: [1],
        3: [0, 4]
        4: [3],
        5: [0, 6],
        6: [5]
    }

'''
visited = []
visited.append(0)
```

```
def BFS(queue):
     # If the queue is empty, the BFS traversal is complete
    if not queue:
        return

    # Dequeue the front node from the queue and print it
    curr_node = queue.pop(0)
    print(curr_node, end=' ')

    # Explore all the neighbors of the current node
    for neighbour in adj_list[curr_node]:
        # If a neighbor has not been visited yet, mark it as visited and enqueue it
        if neighbour not in visited:
            visited.append(neighbour)
            queue.append(neighbour)

    # Recursively call BFS with the updated queue
    BFS(queue)

# Start BFS traversal from the starting node (0) by
# calling BFS with a queue containing only the starting node
BFS([0])
```