Breadth First Search and Depth First Search

1. BFS Code:

In [16]:

```
graph = {
    '5': ['3', '7'],
    '3': ['2', '4'],
    '7': ['8'],
    '2': [],
    '4': ['8'],
    '8': []
}

visited = []
queue = []

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    m = queue.pop(0)
    print(m, end='\n')

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

print("BFS:")
bfs(visited, graph, '5')

# Output:
```

```
BFS:
5
3
7
2
4
8
```

2. DFS Code:

In [18]:

```
graph = {
    '5': ['3', '7'],
    '3': ['2', '4'],
    '7': ['8'],
    '2': [],
    '4': ['8'],
    '8': []
}

visited = []

def dfs(visited, graph, node):
  if node not in visited:
    print(node)
    visited.append(node)
    for neighbour in graph[node]:
      dfs(visited, graph, neighbour)

print("DFS:")
dfs(visited, graph, '5')

# Output:
```

```
DFS:
5
3
2
4
8
7
```

BFS steps:

1. Choose any one node randomly, to start traversing.
2. Visit its adjacent unvisited node.
3. Mark it as visited in the boolean array and display it.
4. Insert the visited node into the queue.
5. If there is no adjacent node, remove the first node from the queue.
6. Repeat the above steps until the queue is empty.

DFS Steps:

1. Create a graph.
2. Initialize a starting node.
3. Send the graph and initial node as parameters to the bfs function.
4. Mark the initial node as visited and push it into the queue.
5. Explore the initial node and add its neighbours to the queue and remove the initial node from the queue.
6. Check if the neighbours node of a neighbouring node is already visited.
7. If not, visit the neighbouring node neighbours and mark them as visited.
8. Repeat this process until all the nodes in a graph are visited and the queue becomes empty.