- **Rule 2:** If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- **Rule 3:** Repeat Rule 1 and Rule 2 until the stack is empty.

SOURCE CODE

```
from Collections import defaultdict
class Graph:
    def __init__(self):
        self.value = defaultdict(list)
    def addConnection(self, parent, child):
        self.value[parent].append(child)
    def DFS(self, start):
        visited = [start]
        stack = [start]
        print(start, end=" ")
        while stack:
            s = stack[-1]
            if any(item for item in self.value[s] if item
                   not in visited):
                for item in (item for item in self.value
                             [s] if item not in visited):
                    stack.append(item)
                    visited(item
                    visited.append(item)
                    print(item, end=" ")
                    break
            else:
                stack.pop()
    def BFS(self, start):
        visited = [start]
```

```
        queue = [start]
        while queue:
            x = queue.pop(0)
            Print (x, end = " ")
            for item in self.value[x]:
                if item not in visited:
                    queue.append(item)
                    Visited.append(item)

g = Graph()
g.addConnection(1,4)
g.addConnection(1,2)
g.addConnection(2,3)
g.addConnection(2,6)
g.addConnection(4,5)
g.addConnection(4,7)
g.addConnection(7,96)
Print("DFS Traversal:")
g.DFS(1)
Print("\nBFS Traversal:")
g.BFS(1)
```

OUTPUT

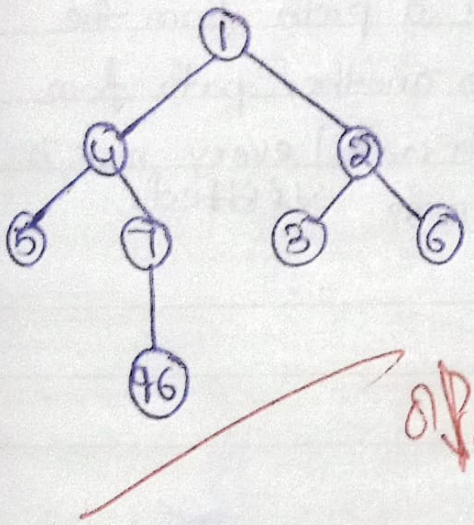
DFS Traversal

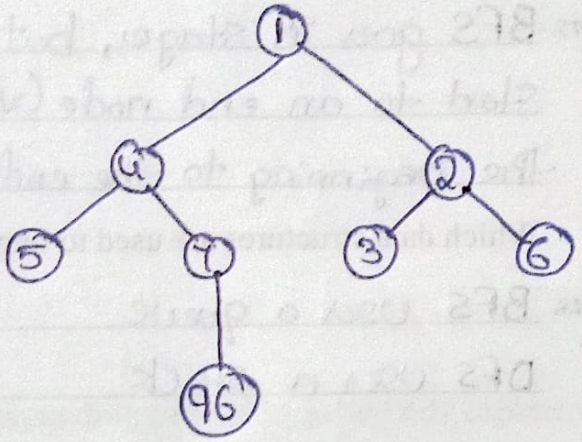
1 4 5 7 96 2 3 6

BFS Traversal

1 4 2 5 7 3 6 96

Graph:-





Result:-

Using IDLE (Python 3.9.6 - 64-bit) we have successfully executed DFS & BFS Program

VIVA QUESTIONS

1. What is a Graph and tree?

Ans. Graphs are networks consisting of nodes connected by edges (or) arcs. Trees are non-linear data structures that store data hierarchically & are made up of nodes connected by edges.

2. What are the main differences between BFS and DFS?

Ans. BFS goes in stages, but DFS takes a path from the start to an end node (vertex), then another path from the beginning to the end, & so forth until every node is visited.

3. Which data structures are used to implement BFS and DFS?

Ans. BFS uses a queue

DFS uses a stack

4. What is the space complexity of BFS and DFS?

Ans. DFS is O(V), where V represents the number of vertices in the graph. & for BFS it is O(V), where V represents the number of vertices in the graph.

5. How does BFS handle cycles in a graph?

Ans. Initialise an empty queue. This queue will be holding a pair, node and the previous node.

Week-1

Aim:-

Write a program to implement BFS and DFS Traversal

Algorithm : BFS :-

→ Each vertex or node in graph is known. For instance, you can mark the node as V.

→ In case the vertex V is not accessed then add the vertex V into the BFS Queue.

→ Start the BFS search, and after completion, mark vertex V as visited.

→ The BFS queue is still not empty, hence remove the vertex V of the graph from the queue.

→ Retrieve all the remaining vertices on the graph that are adjacent to the vertex V.

→ For each adjacent vertex let's say V_1, in case it is not visited yet then add V_1 to the BFS queue

→ BFS will visit V_1 and mark it as visited and delete it from the queue.

Algorithm : DFS

Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

→ Rule 1: Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

→ Rule 2: If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)

→ Rule 3: Repeat Rule 1 and Rule 2 until the stack is empty.

Source Code :-

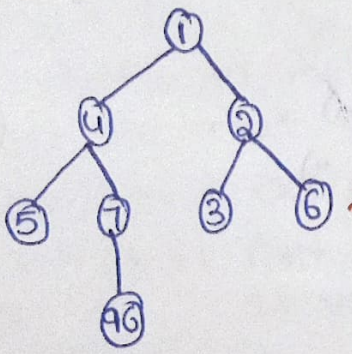
```
from Collections import defaultdict
class Graph:
    def __init__(self):
        self.graph = defaultdict(list)
    def addConnection(Self, Parent, child):
        Self.Value[Parent].append(child)
    def DFS(Self, start):
        Visited = [start]
        Stark = [start]
        Print(start, end=" ")
        while stack:
            s = stack[-1]
            if any(item for item in self.value[s] if
                                item not in visited):
                for item in [item for item in self.values
                                [s]
                            if item not in visited]:
                    stack.append(item)
                    visited.append(item)
                    Print(item, end=" ")
                    break
                else:
                    stack.pop()

    def BFS(Self, start):
        Visited = [start]
        queue = [start]
        while queue:
            X = queue.pop(0)
            Print(X, end=" ")
            for item in self.value[x]:
                queue.append(item)
                visited.append(item)
```

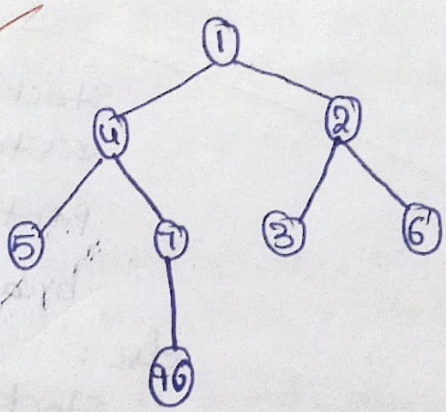
```
g= Graph()
g.addConnection(1,4)
g.addConnection(1,2)
g.addConnection(2,3)
g.add Connection(2,6)
g.add Connection(4,5)
g.add Connection(4,7)
g.addConnection(7,96)
Print("DFS. Traversal:")
g.DFS(1)
Print("\n Bfs Traversal:")
g. BFS(1)
```

Graph:-

DFS:-



BFS:-



Output:-

DFs Traversal:

1 4 5 7 96 2 3 6

BFs Traversal:

1 4 2 5 7 3 6 96

Result:-

Using IDLE (Python 3.9.64-bit) we have successfully executed DFS & BFS program.