```
from queue import Queue

class Graph:
    n = None
    matrix = []
    dfs_list = []
    bfs_list = []
    def __init__(self, matrix: list, verbose: bool = False, n: int = 0) -> None:
        """
        Args:
            matrix (list): adjency matrix
            verbose (bool, optional): To use terminal. Defaults to False.
            n (int, optional): number of nodes. Defaults to 0.
        """
        if verbose:
            for i in range(n):
                row = []
                for j in range(n):
                    connection = int(
                        input("Enter 0 if no connection between " + str(i) + "
and " + str(j) + " else enter 1: "))
                    row.append(connection)
                self.matrix.append(row)
        else:
            self.matrix = matrix
    def printGraph(self) -> None:
        """
        Function for displaying graph
        """
        for row in self.matrix:
            print(row)
    def dfs(self, source: int, visited: list) -> list:
        """
        DFS implementation using adjacency matrix
        Args:
            source (int): starting node of the graph
            visited (list): boolean list for visited nodes
        Returns:
            list: dfs sequence
        """
        self.dfs_list.append(source)
        visited[source] = True
        row = self.matrix[source]
        for node in range(len(row)):
            if (visited[node] == False and self.matrix[source][node] != 0):
                self.dfs(node, visited)
        return self.dfs_list
    def bfs(self, source: int, visited: list) -> list:
        """
        BFS implementation using adjacency matrix
        Args:
            source (int): starting node of the graph
            visited (list): boolean list for visited nodes
        Returns:
            list: bfs sequence
        """
        queue = Queue()
        queue.put(source)
        visited[source] = True
        while (queue.empty() != True):
            curr = queue.get()
            self.bfs_list.append(curr)
            row = self.matrix[curr]
            for node in range(len(row)):
```

```
                if (visited[node] == False and self.matrix[curr][node] != 0):
                    queue.put(node)
                    visited[node] = True
        return self.bfs_list
    def to_file(self, filename: str, language: str = "py") -> bool:
        f = open(filename, 'w')
        code = """from queue import Queue
        class Graph:
            n = None
            matrix = []
            dfs_list = []
            bfs_list = []
            def __init__(self, matrix:list, verbose:bool=False, n:int=0) ->
None:
                if verbose:
                    for i in range(n):
                        row = []
                        for j in range(n):
                            connection = int(input("Enter 0 if no connection
between " + str(i) + " and " + str(j) + " else enter 1: "))
                            row.append(connection)
                        self.matrix.append(row)
                else:
                    self.matrix = matrix
            def printGraph(self) -> None:
                for row in self.matrix:
                    print(row)
            def dfs(self, source:int, visited:list) -> list:
                self.dfs_list.append(source)
                visited[source] = True
                row = self.matrix[source]
                for node in range(len(row)):
                    if (visited[node] == False and self.matrix[source][node] !=
0):
                        self.dfs(node, visited)
                return self.dfs_list
            def bfs(self, source:int, visited:list) -> list:
                queue = Queue()
                queue.put(source)
                visited[source] = True
                while (queue.empty() != True):
                    curr = queue.get()
                    self.bfs_list.append(curr)
                    row = self.matrix[curr]
                    for node in range(len(row)):
                        if (visited[node] == False and self.matrix[curr][node] !
= 0):
                            queue.put(node)
                            visited[node] = True
                return self.bfs_list
        """
        f.write(code)
        f.close()
        return True


if __name__ == "__main__":
    matrix = [[0, 1, 1, 0],
              [1, 0, 0, 1],
              [1, 0, 0, 1],
              [1, 1, 1, 0]]
    obj = Graph(matrix)
    obj.printGraph()
    visited = [False] * 4
```

```
    print(obj.dfs(0, visited))
    visited = [False] * 4
    print(obj.bfs(0, visited))
    obj.to_file("code.py")

/*
output:-
[0, 1, 1, 0]
[1, 0, 0, 1]
[1, 0, 0, 1]
[1, 1, 1, 0]
[0, 1, 3, 2]
[0, 1, 2, 3]  */
```