EXPERIMENT 1
BFS & DFS

AIM	**WAP to implement DFS and BFS for traversing a graph from source node (S) to goal node (G), where source node and goal node is given by the user as an input.**

CODE

```
  # Input: Adjacency matrix and source/goal nodes
def graph_input():
    d_mtx = []
    n = int(input("Enter the number of nodes in the graph: "))
    print("Enter the existence of edges between nodes (1 for edge,
0 for no edge):")

    # Building the adjacency matrix



    for i in range(n):
        d_mtx.append([])
        for j in range(n):
            d_mtx[-1].append(int(input(f"Edge between node {i} and
node {j}: ")))

    s = int(input("Enter the index number of the source node: "))
    g = int(input("Enter the index number of the goal node: "))
    return d_mtx, n, s, g

# BFS Algorithm
def bfs_algorithm(d_mtx, n, s, g):
    print("\nRunning BFS...")
    queue = [s]  # Queue to store nodes to visit
    visited = [s]  # List to keep track of visited nodes
    paths = [[s]]  # List to store paths

    if s == g:
```

```
        print(f"Goal node {g} found in BFS from {s}")
        print(f"Path traversed: {paths[0]}")
        return

    found = False
    while queue:
        node = queue.pop(0)
        path = paths.pop(0)

        for i in range(n):
            if d_mtx[node][i] == 1:  # Check if there's an edge
                if i == g:  # Goal found
                    print(f"Goal node {i} found in BFS from {s}")
                    print(f"Traversal path found using BFS: {path +
[i]}")
                    print(f"Nodes checked in BFS: {visited}")
                    found = True
                    break
                if i not in visited:  # If not visited
                    queue.append(i)
                    visited.append(i)
                    paths.append(path + [i])

        if found:
            break

    if not found:
        print(f"Goal node {g} not found in BFS from {s}")

# DFS Algorithm
def dfs_algorithm(d_mtx, n, s, g):
    print("\nRunning DFS...")
    stack = [s]  # Stack to store nodes to visit
    visited = [s]  # List to keep track of visited nodes

    if s == g:
        print(f"Goal node {g} found in DFS from {s}")
        print(f"Path traversed: {stack + [g]}")
        return
```

```
    found = False
    while stack:
        node = stack[-1]
        unvisited_found = False

        for i in range(n):
            if d_mtx[node][i] == 1:  # Check if there's an edge
                if i == g:  # Goal found
                    print(f"Goal node {i} found in DFS from {s}")
                    print(f"Traversal path found using DFS: {stack +
[i]}")
                    print(f"Nodes checked in DFS: {visited}")
                    found = True
                    break
                if i not in visited:  # If not visited
                    stack.append(i)
                    visited.append(i)
                    unvisited_found = True
                    break

        if not unvisited_found:  # No unvisited neighbors, backtrack
            stack.pop()

        if found:
            break

    if not found:
        print(f"Goal node {g} not found in DFS from {s}")

# Main function
if __name__ == "__main__":
    d_mtx, n, s, g = graph_input()

    # Validate source and goal nodes
    if 0 <= s < n and 0 <= g < n:
        bfs_algorithm(d_mtx, n, s, g)
        dfs_algorithm(d_mtx, n, s, g)
    else:
```

	print(f"Invalid node indices. The values must be in the range [0, {n-1}]")

OUTPUT

```
Enter the number of nodes in the graph: 5
Enter the existence of edges between nodes (1 for edge, 0 for no edge):
Edge between node 0 and node 0: 0
Edge between node 0 and node 1: 1
Edge between node 0 and node 2: 1
Edge between node 0 and node 3: 1
Edge between node 0 and node 4: 1
Edge between node 1 and node 0: 0
Edge between node 1 and node 1: 1
Edge between node 1 and node 2: 0
Edge between node 1 and node 3: 1
Edge between node 1 and node 4: 0
Edge between node 2 and node 0: 1
Edge between node 2 and node 1: 0
Edge between node 2 and node 2: 1
Edge between node 2 and node 3: 1
Edge between node 2 and node 4: 1
Edge between node 3 and node 0: 0
Edge between node 3 and node 1: 1
Edge between node 3 and node 2: 0
Edge between node 3 and node 3: 0
Edge between node 3 and node 4: 0
Edge between node 4 and node 0: 1
Edge between node 4 and node 1: 1
Edge between node 4 and node 2: 1
```

```
Running BFS...
Goal node 2 found in BFS from 1
Traversal path found using BFS: [1, 2]
Nodes checked in BFS: [1, 0]

Running DFS...
Goal node 2 found in DFS from 1
Traversal path found using DFS: [1, 0, 2]
Nodes checked in DFS: [1, 0]
```

	VISUAL REPRESENTATION