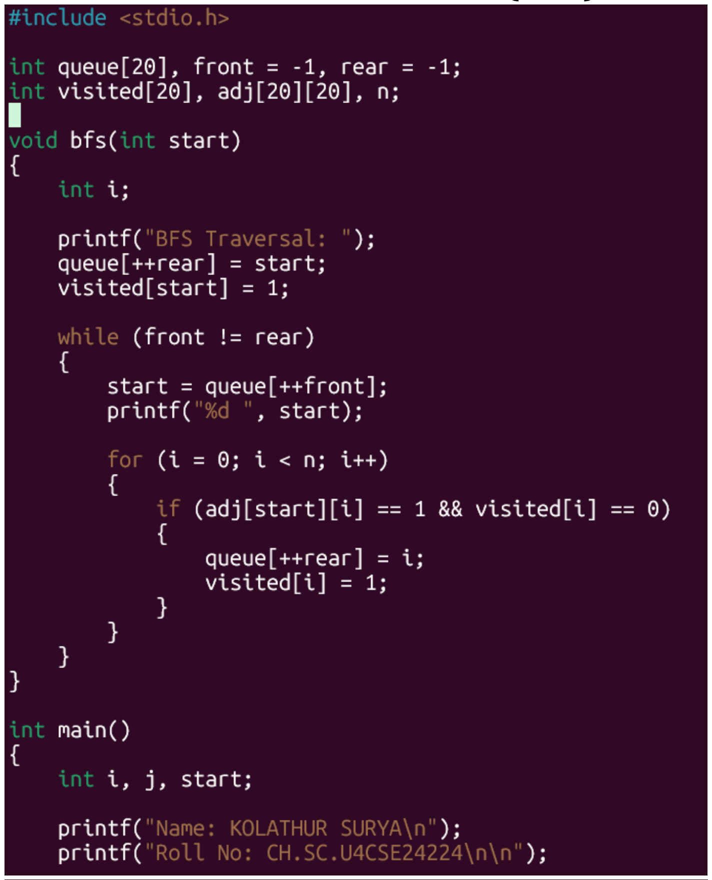
2025

DESINGN AND ANALYSIS OF ALGORITHM

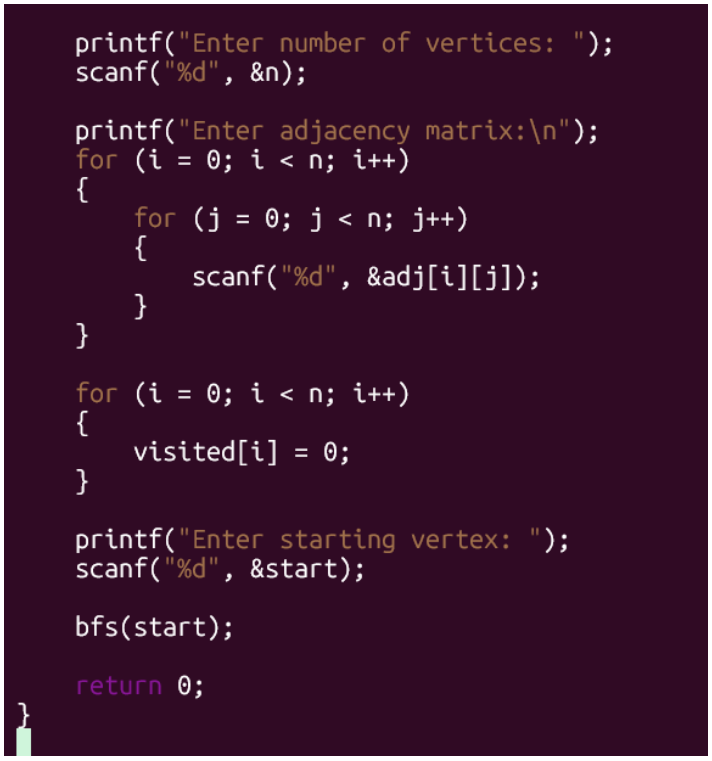
LAB EXPERIMENT -3

KOLATHUR SURYA
[CH.SC.U4CSE24224]

1. BREADTH – FIRST SEARCH(BFS) Code:

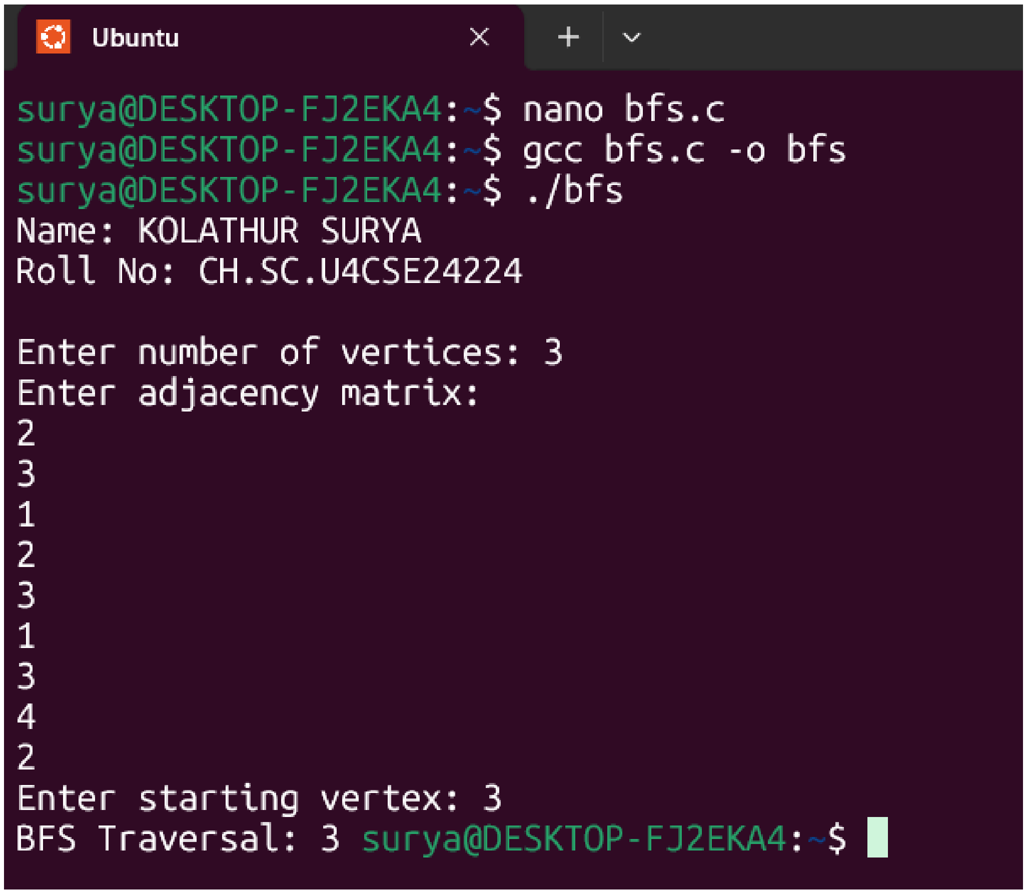


```c
#include <stdio.h>

int queue[20], front = -1, rear = -1;
int visited[20], adj[20][20], n;

void bfs(int start)
{
    int i;

    printf("BFS Traversal: ");
    queue[++rear] = start;
    visited[start] = 1;

    while (front != rear)
    {
        start = queue[++front];
        printf("%d ", start);

        for (i = 0; i < n; i++)
        {
            if (adj[start][i] == 1 && visited[i] == 0)
            {
                queue[++rear] = i;
                visited[i] = 1;
            }
        }
    }
}

int main()
{
    int i, j, start;

    printf("Name: KOLATHUR SURYA\n");
    printf("Roll No: CH.SC.U4CSE24224\n\n");
```



```c
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            scanf("%d", &adj[i][j]);
        }
    }

    for (i = 0; i < n; i++)
    {
        visited[i] = 0;
    }

    printf("Enter starting vertex: ");
    scanf("%d", &start);

    bfs(start);

    return 0;
}
```

Output:



```
surya@DESKTOP-FJ2EKA4:~$ nano bfs.c
surya@DESKTOP-FJ2EKA4:~$ gcc bfs.c -o bfs
surya@DESKTOP-FJ2EKA4:~$ ./bfs
Name: KOLATHUR SURYA
Roll No: CH.SC.U4CSE24224

Enter number of vertices: 3
Enter adjacency matrix:
2
3
1
2
3
1
3
4
2
Enter starting vertex: 3
BFS Traversal: 3 surya@DESKTOP-FJ2EKA4:~$
```

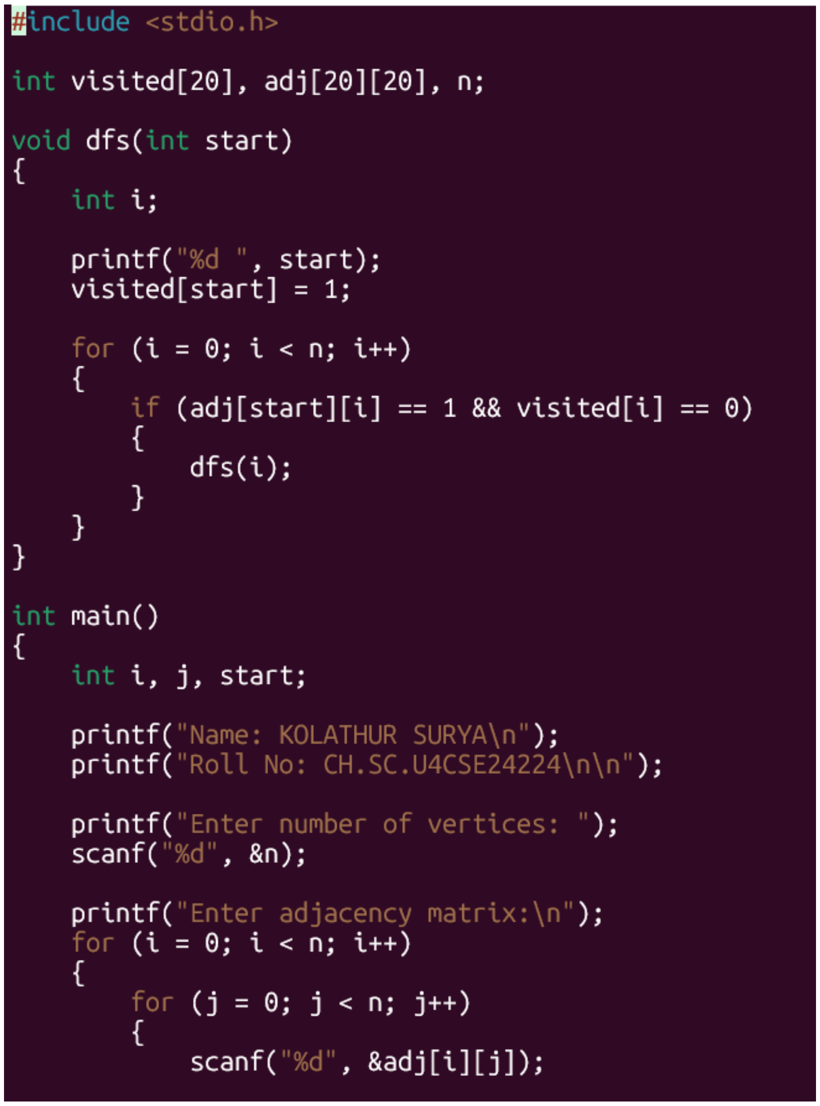
- **Time Complexity: O(N^2)**
  Using an adjacency matrix, each vertex checks all N possible neighbors. Repeating this for N vertices gives O(N^2) time.

- **Space Complexity: O(N)**

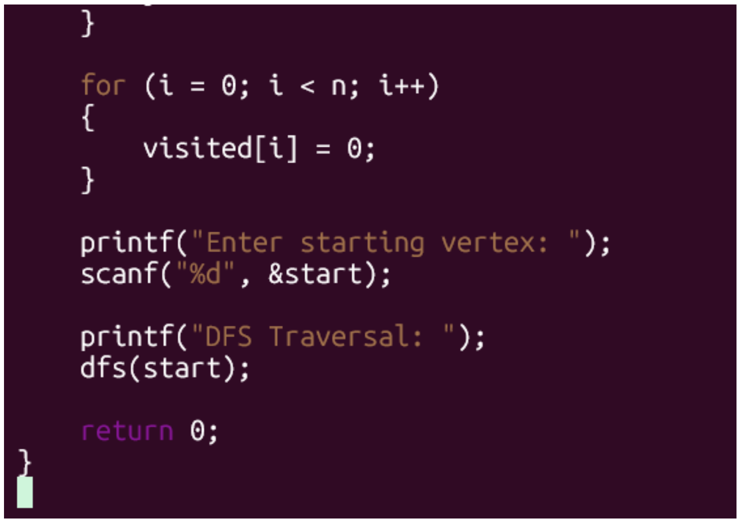
The queue can store up to N vertices in the worst case, so the space complexity is O(N).

2. DEPTH – FIRST SEARCH(DFS)

Code:

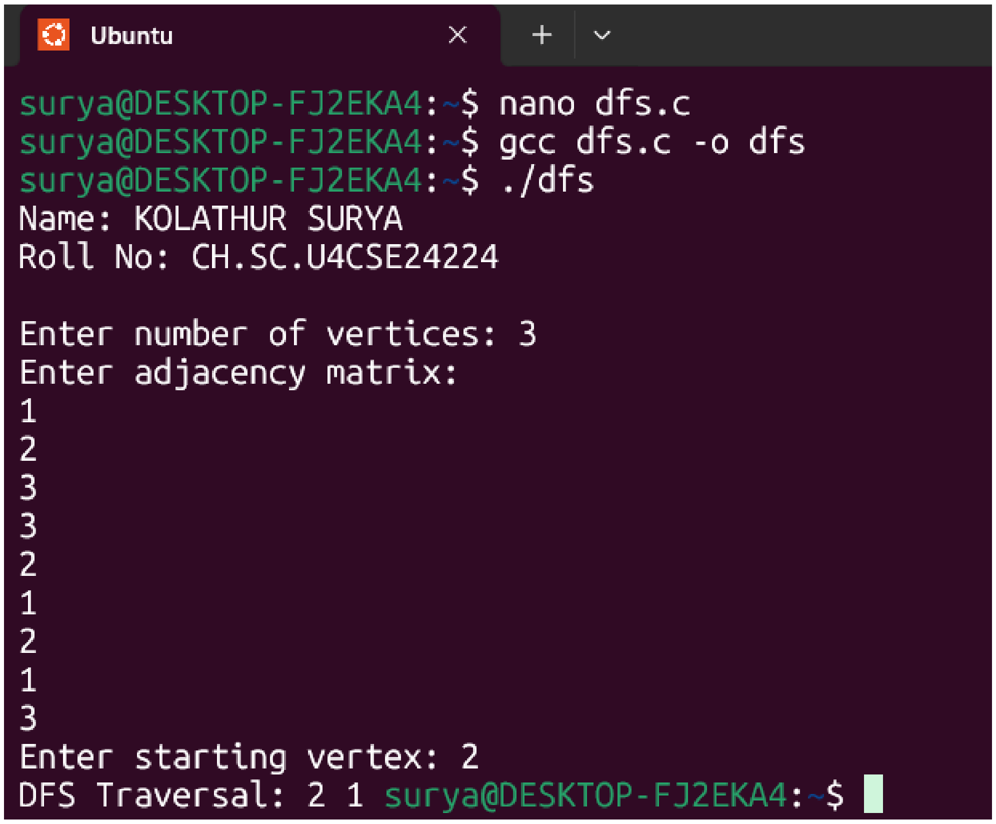


```c
#include <stdio.h>

int visited[20], adj[20][20], n;

void dfs(int start)
{
    int i;

    printf("%d ", start);
    visited[start] = 1;

    for (i = 0; i < n; i++)
    {
        if (adj[start][i] == 1 && visited[i] == 0)
        {
            dfs(i);
        }
    }
}

int main()
{
    int i, j, start;

    printf("Name: KOLATHUR SURYA\n");
    printf("Roll No: CH.SC.U4CSE24224\n\n");

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            scanf("%d", &adj[i][j]);
```



```c
    }

    for (i = 0; i < n; i++)
    {
        visited[i] = 0;
    }

    printf("Enter starting vertex: ");
    scanf("%d", &start);

    printf("DFS Traversal: ");
    dfs(start);

    return 0;
}
```

Output:



```
surya@DESKTOP-FJ2EKA4:~$ nano dfs.c
surya@DESKTOP-FJ2EKA4:~$ gcc dfs.c -o dfs
surya@DESKTOP-FJ2EKA4:~$ ./dfs
Name: KOLATHUR SURYA
Roll No: CH.SC.U4CSE24224

Enter number of vertices: 3
Enter adjacency matrix:
1
2
3
3
2
1
2
1
3
Enter starting vertex: 2
DFS Traversal: 2 1 surya@DESKTOP-FJ2EKA4:~$
```

- **Time Complexity: O(N^2)**

DFS uses an adjacency matrix, so for each vertex it scans all N vertices to find neighbors. Hence, the total time complexity is O(N^2).

- **Space Complexity: O(N)**

In the worst case, DFS recursion can go up to N levels, requiring O(N) stack space.