



SCHOOL OF
COMPUTING

M M VASANTH

CH.SC.U4CSE24227

Week - 3

Date - 19/12/2025

Design and Analysis of Algorithm(23CSE211)

1. BREADTH – FIRST SEARCH(BFS)

Code:

```
#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: M M VASANTH\n");
    printf("Roll No: CH.SC.U4CSE24227\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]);
    for (i = 0; i < n; i++)
```

```

        visited[i] = 0;
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    bfs(start);
    return 0;
}

```

Output:

```

amma@amma24: ~
amma@amma24:~$ nano bfs.c
amma@amma24:~$ gcc bfs.c -o bfs
./bfs
Name: M M VASANTH
Roll No: CH.SC.U4CSE24227

Enter number of vertices: 3
Enter adjacency matrix:
2
5
7
9
5
3
7
9
97
Enter starting vertex: 5
BFS Traversal: 5 amma@amma24:~$

```

☑ Time Complexity: $O(N^2)$

Since the graph is represented using an Adjacency Matrix (`adj[n][n]`), for every vertex we visit (dequeued), we must iterate through all N potential neighbors in the inner for loop to check for connections. Doing this for all N vertices results in $N * N$ operations.

☑ Space Complexity: $O(N)$

The queue array stores the vertices to be visited. In the worst case, the queue might store all vertices of the graph, making the space complexity proportional to the number of vertices N .

2. DEPTH – FIRST SEARCH(DFS)

Code:

```
#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: M M VASANTH\n");
    printf("Roll No: CH.SC.U4CSE24227\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]);
    for (i = 0; i < n; i++)
```

```
        visited[i] = 0;

    printf("Enter starting vertex: ");

    scanf("%d", &start);

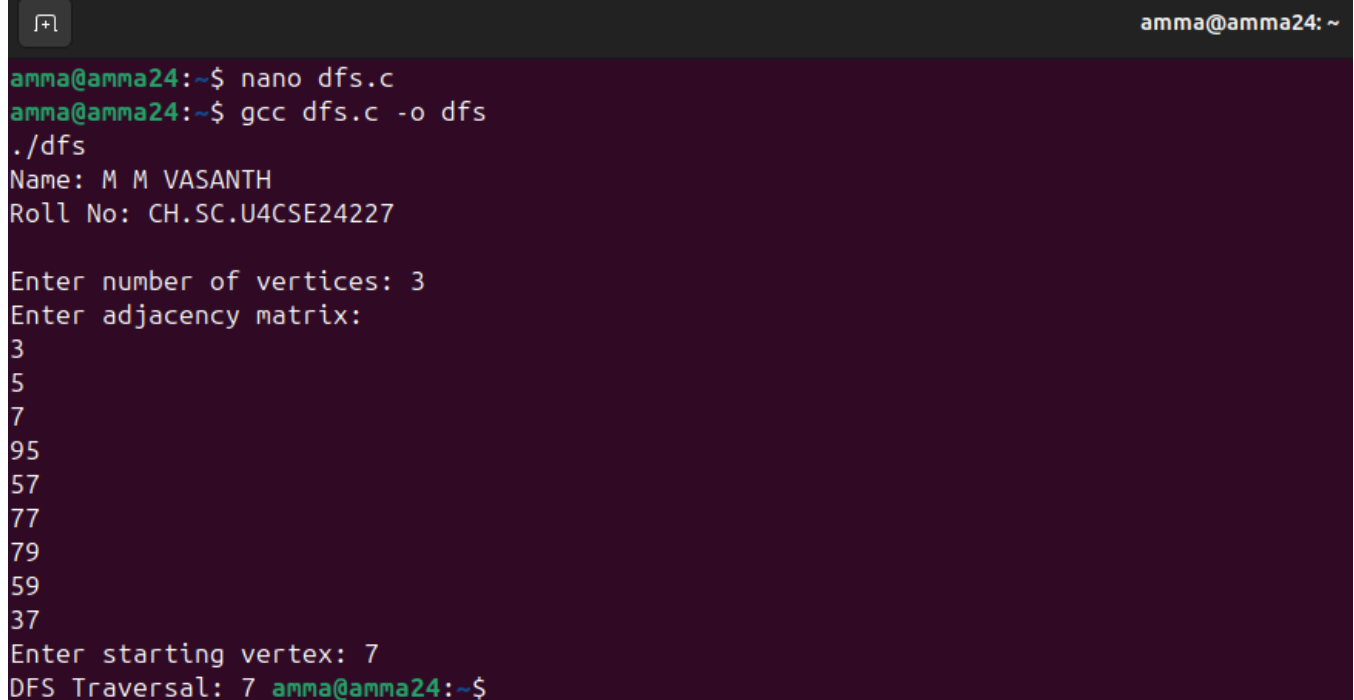
    printf("DFS Traversal: ");

    dfs(start);

    return 0;

}
```

Output:



A terminal window with a dark purple background and light green text. The window title is 'amma@amma24: ~'. The user has executed the following commands: 'nano dfs.c', 'gcc dfs.c -o dfs', and './dfs'. The program output includes a name 'M M VASANTH', a roll number 'CH.SC.U4CSE24227', and prompts for the number of vertices (3) and the starting vertex (7). The final output is 'DFS Traversal: 7'.

```
amma@amma24:~$ nano dfs.c
amma@amma24:~$ gcc dfs.c -o dfs
./dfs
Name: M M VASANTH
Roll No: CH.SC.U4CSE24227

Enter number of vertices: 3
Enter adjacency matrix:
3
5
7
95
57
77
79
59
37
Enter starting vertex: 7
DFS Traversal: 7 amma@amma24:~$
```

⌘ Time Complexity: $O(N^2)$

Similar to BFS, this implementation uses an Adjacency Matrix. The recursive function is called for every vertex. Inside each call, the for loop scans the entire row of the matrix (size N) to find adjacent nodes. Thus, the total time complexity is $O(N^2)$.

⌘ Space Complexity: $O(N)$

The space is used by the Recursion Stack. In the worst-case scenario (e.g., a graph that is a long straight line), the recursion depth can go up to N , occupying $O(N)$ stack memory.