Lab -

DIP : Design and Analysis of Algorithm

AIM : Write a program to solve the following Algorithm

1. Breadth First Search.

2. Depth First Search.

Name : Ambalia Harshit

Department : Computer Engineering (M.Tech sem II)

Roll no : MT001

Date : Jan 2024

Program 01 : Depth First Search(Without recursion)

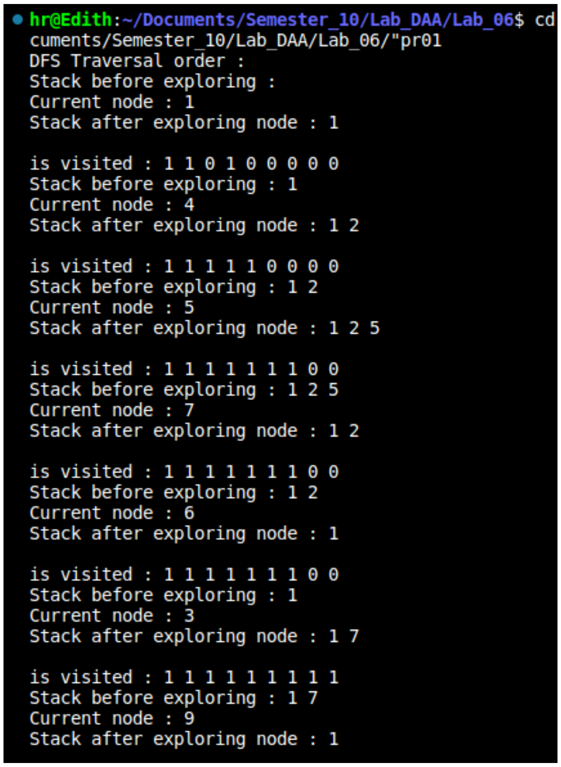
- Code :

```
// DFS

#include <stdio.h>

void print_1d_integer_array(int integer_array[], int n) {
    for( int i=0;i<n;i++ ) {
        printf("%d ", integer_array[i]);
    }
    printf("\n");
}

void dfs(int start, int length, int graph[length][length], int visited[length], int parent[length]) {
    int stack[(length*(length-1))/2]; // Considering mesh topology - Max edge count is : (length*(length-1))/2
    int top = -1;

    top++;
    stack[top] = start;
    visited[start] = 1;

    printf("DFS Traversal order :\n");
    while( top!=-1 ) {

        printf("Stack before exploring : ");
        print_1d_integer_array(stack, top);

        // Visiting each node : O(n)
        int current = stack[top];
        top--;
        printf("Current node : %d ", current+1);

        // Exploring adjacent nodes of current
        for( int i=0;i<length;i++ ) {
            // Visiting adjucent nodes for each vertices : O()
            if (graph[current][i] && !visited[i]) {
```

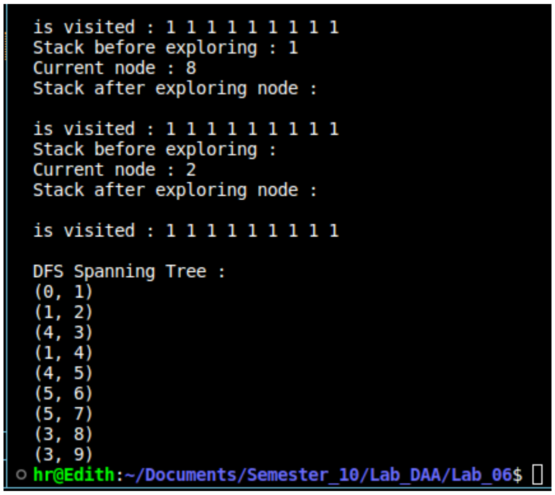
```c
                top++;
                stack[top] = i; // Add adjecent nodes to stack
                visited[i] = 1; // Node is now visited
                parent[i] = current; // As we reached from curent to this node
            }
        }
        printf("\nStack after exploring node : ");
        print_1d_integer_array(stack, top);
        printf("\nis visited : ");
        print_1d_integer_array(visited, length);
    }

    printf("\nDFS Spanning Tree :\n");
    for( int i=0;i<length;i++ ) {
        printf("(%d, %d)\n", parent[i]+1, i+1);
    }
}

int main() {
    int n=9;

    int graph[9][9] = {
        {0, 1, 0, 1, 0, 0, 0, 0, 0},
        {1, 0, 1, 0, 0, 0, 0, 0, 0},
        {0, 1, 0, 1, 0, 0, 0, 1, 1},
        {1, 0, 1, 0, 1, 0, 0, 0, 0},
        {0, 0, 0, 1, 0, 1, 1, 0, 0},
        {0, 0, 0, 0, 1, 0, 0, 0, 0},
        {0, 0, 0, 0, 1, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 0, 0, 0},
    };
    int visited[9] = {0};
    int parent[9] = {-1};
    int start = 0;

    dfs(start, n, graph, visited, parent);

    return 0;
```

```
}

/*
   Time complexity :
   Visiting n vertex once : O(n)
   For each vertex, exploring its m adjacent nodes(edges) : O(n+m)
*/
```

- Output Screen-shots :



```
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_06$ cd
cuments/Semester_10/Lab_DAA/Lab_06/"pr01
DFS Traversal order :
Stack before exploring :
Current node : 1
Stack after exploring node : 1

is visited : 1 1 0 1 0 0 0 0 0
Stack before exploring : 1
Current node : 4
Stack after exploring node : 1 2

is visited : 1 1 1 1 1 0 0 0 0
Stack before exploring : 1 2
Current node : 5
Stack after exploring node : 1 2 5

is visited : 1 1 1 1 1 1 1 0 0
Stack before exploring : 1 2 5
Current node : 7
Stack after exploring node : 1 2

is visited : 1 1 1 1 1 1 1 0 0
Stack before exploring : 1 2
Current node : 6
Stack after exploring node : 1

is visited : 1 1 1 1 1 1 1 0 0
Stack before exploring : 1
Current node : 3
Stack after exploring node : 1 7

is visited : 1 1 1 1 1 1 1 1 1
Stack before exploring : 1 7
Current node : 9
Stack after exploring node : 1
```

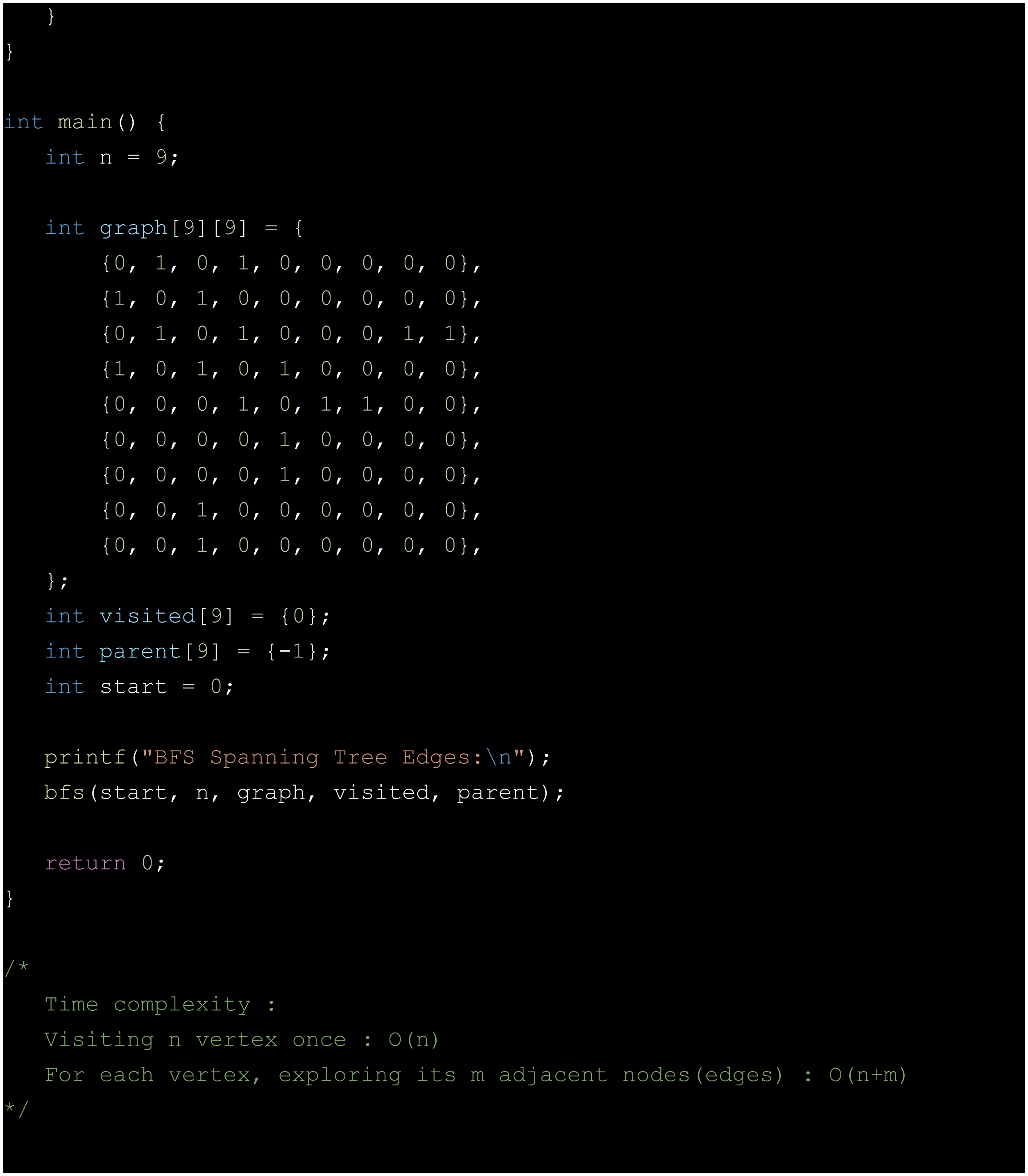


```
is visited : 1 1 1 1 1 1 1 1 1
Stack before exploring : 1
Current node : 8
Stack after exploring node :

is visited : 1 1 1 1 1 1 1 1 1
Stack before exploring :
Current node : 2
Stack after exploring node :

is visited : 1 1 1 1 1 1 1 1 1

DFS Spanning Tree :
(0, 1)
(1, 2)
(4, 3)
(1, 4)
(4, 5)
(5, 6)
(5, 7)
(3, 8)
(3, 9)
hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_06$
```

Program 02 : Breadth First Search.

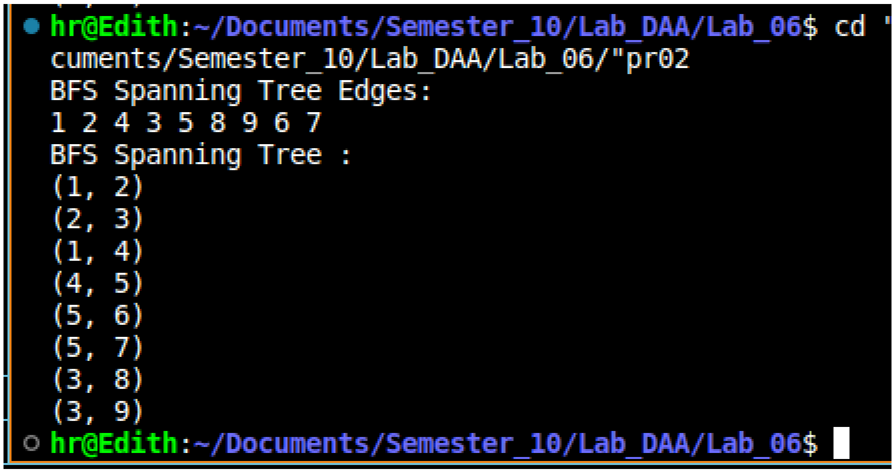
- Code :

```
// BFS

#include <stdio.h>

void bfs(int start, int length, int graph[length][length], int
visited[length], int parent[length]) {
   int queue[(length*(length-1))/2]; // Considering mesh topology - Max
edge count is : (length*(length-1))/2
   int front = 0;
   int rear = -1;

   rear++;
   queue[rear] = start;
   visited[start] = 1;

   while( front<=rear ) {
       int current = queue[front++];
       printf("%d ", current+1);

       // Explore adjacent nodes of current
       for( int i=0;i<length;i++ ) {
           if( graph[current][i] && !visited[i] ) {
               rear++;
               queue[rear] = i; // Add adjacent nodes to the queue
               visited[i] = 1; // Node is now visited
               parent[i] = current;// As we reached from curent to this
node
           }
       }
   }

   printf("\nBFS Spanning Tree :\n");
   for( int i=0;i<length;i++ ) {
       if( parent[i]!=-1 ) {
           printf("(%d, %d)\n", parent[i]+1, i+1);
       }
```



```
    }
}

int main() {
    int n = 9;

    int graph[9][9] = {
        {0, 1, 0, 1, 0, 0, 0, 0, 0},
        {1, 0, 1, 0, 0, 0, 0, 0, 0},
        {0, 1, 0, 1, 0, 0, 0, 1, 1},
        {1, 0, 1, 0, 1, 0, 0, 0, 0},
        {0, 0, 0, 1, 0, 1, 1, 0, 0},
        {0, 0, 0, 0, 1, 0, 0, 0, 0},
        {0, 0, 0, 0, 1, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 0, 0, 0},
    };
    int visited[9] = {0};
    int parent[9] = {-1};
    int start = 0;

    printf("BFS Spanning Tree Edges:\n");
    bfs(start, n, graph, visited, parent);

    return 0;
}

/*
    Time complexity :
    Visiting n vertex once : O(n)
    For each vertex, exploring its m adjacent nodes(edges) : O(n+m)
*/
```

- Output Screen-Shots :

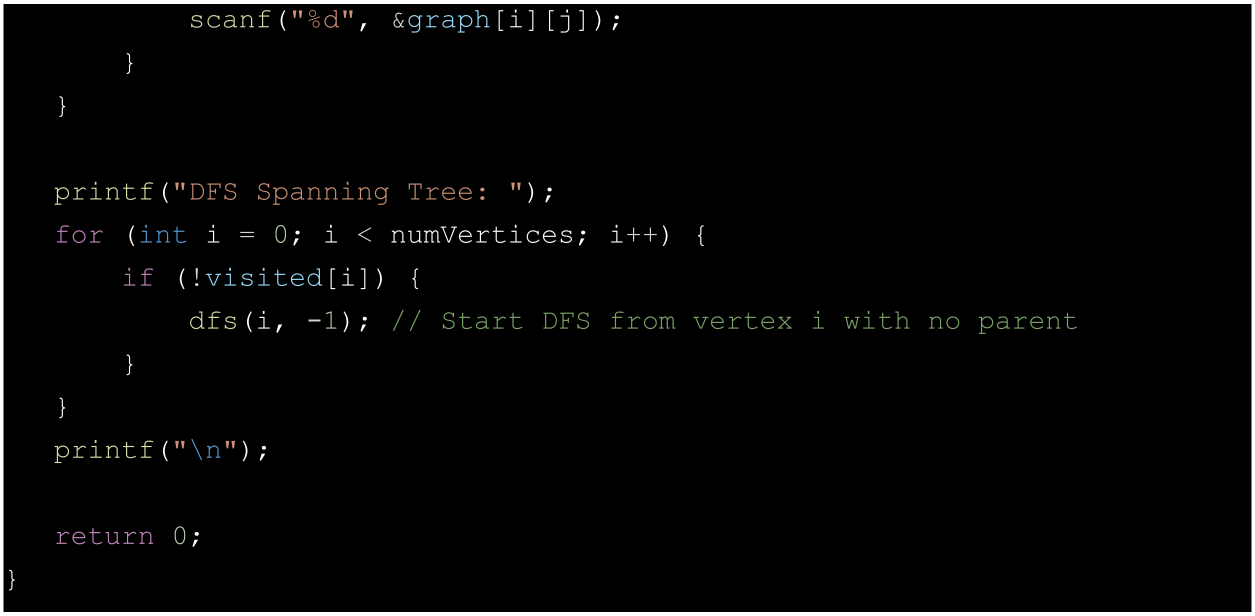


```
● hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_06$ cd '
  cuments/Semester_10/Lab_DAA/Lab_06/"pr02
  BFS Spanning Tree Edges:
  1 2 4 3 5 8 9 6 7
  BFS Spanning Tree :
  (1, 2)
  (2, 3)
  (1, 4)
  (4, 5)
  (5, 6)
  (5, 7)
  (3, 8)
  (3, 9)
○ hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_06$
```

Extra 01 : Depth First Search(Using recursion)

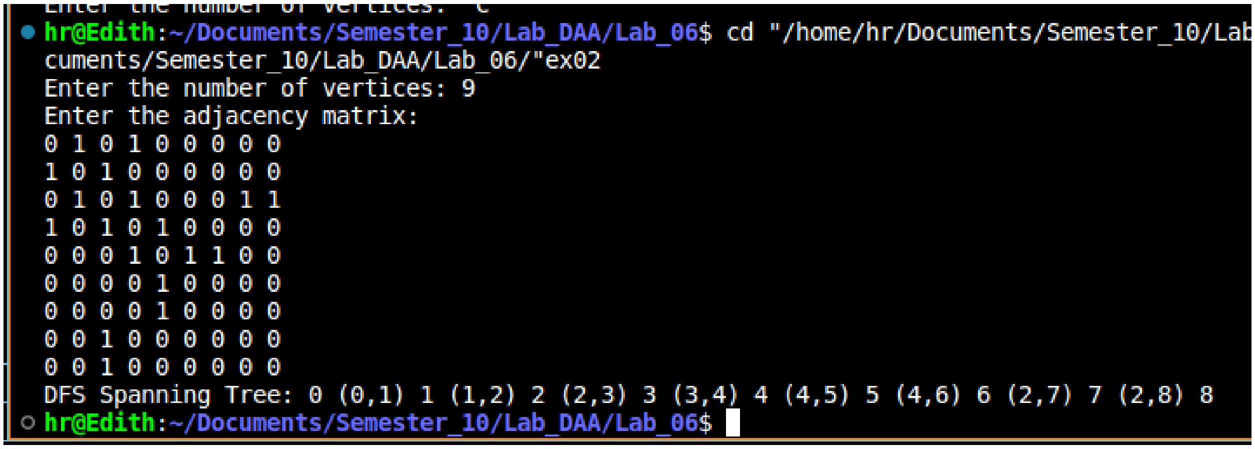
- Code :

```
// DFS
#include <stdio.h>

#define MAX_VERTICES 100

int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];
int numVertices;

void dfs(int vertex, int parent) {
    visited[vertex] = 1;
    printf("%d ", vertex);

    for (int i = 0; i < numVertices; i++) {
        if (graph[vertex][i] && !visited[i]) {
            printf("(%d,%d) ", vertex, i); // Print edge (parent, i)
            dfs(i, vertex); // Recursively visit i
        }
    }
}

int main() {
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    // Initialize graph and visited array
    for (int i = 0; i < numVertices; i++) {
        visited[i] = 0;
        for (int j = 0; j < numVertices; j++) {
            graph[i][j] = 0;
        }
    }

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
```



```
            scanf("%d", &graph[i][j]);
        }
    }

    printf("DFS Spanning Tree: ");
    for (int i = 0; i < numVertices; i++) {
        if (!visited[i]) {
            dfs(i, -1); // Start DFS from vertex i with no parent
        }
    }
    printf("\n");

    return 0;
}
```

- Output Screen-Shots :

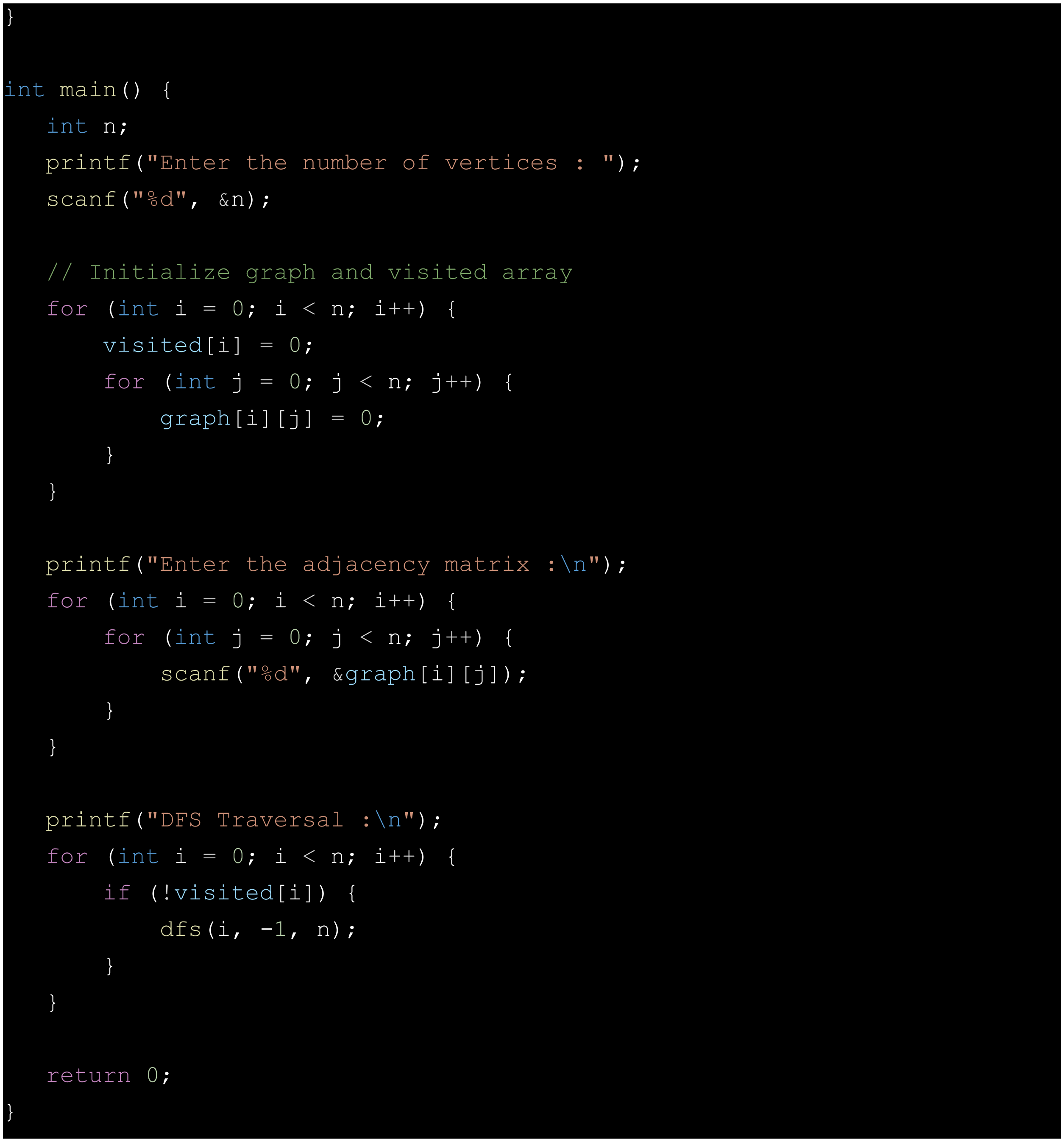


```
Enter the number of vertices: ^C
● hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_06$ cd "/home/hr/Documents/Semester_10/Lab
cuments/Semester_10/Lab_DAA/Lab_06/"ex02
Enter the number of vertices: 9
Enter the adjacency matrix:
0 1 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0 0
0 1 0 1 0 0 0 1 1
1 0 1 0 1 0 0 0 0
0 0 0 1 0 1 1 0 0
0 0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0
DFS Spanning Tree: 0 (0,1) 1 (1,2) 2 (2,3) 3 (3,4) 4 (4,5) 5 (4,6) 6 (2,7) 7 (2,8) 8
○ hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_06$ 
```

Extra 02 : Depth First Search(all edges)

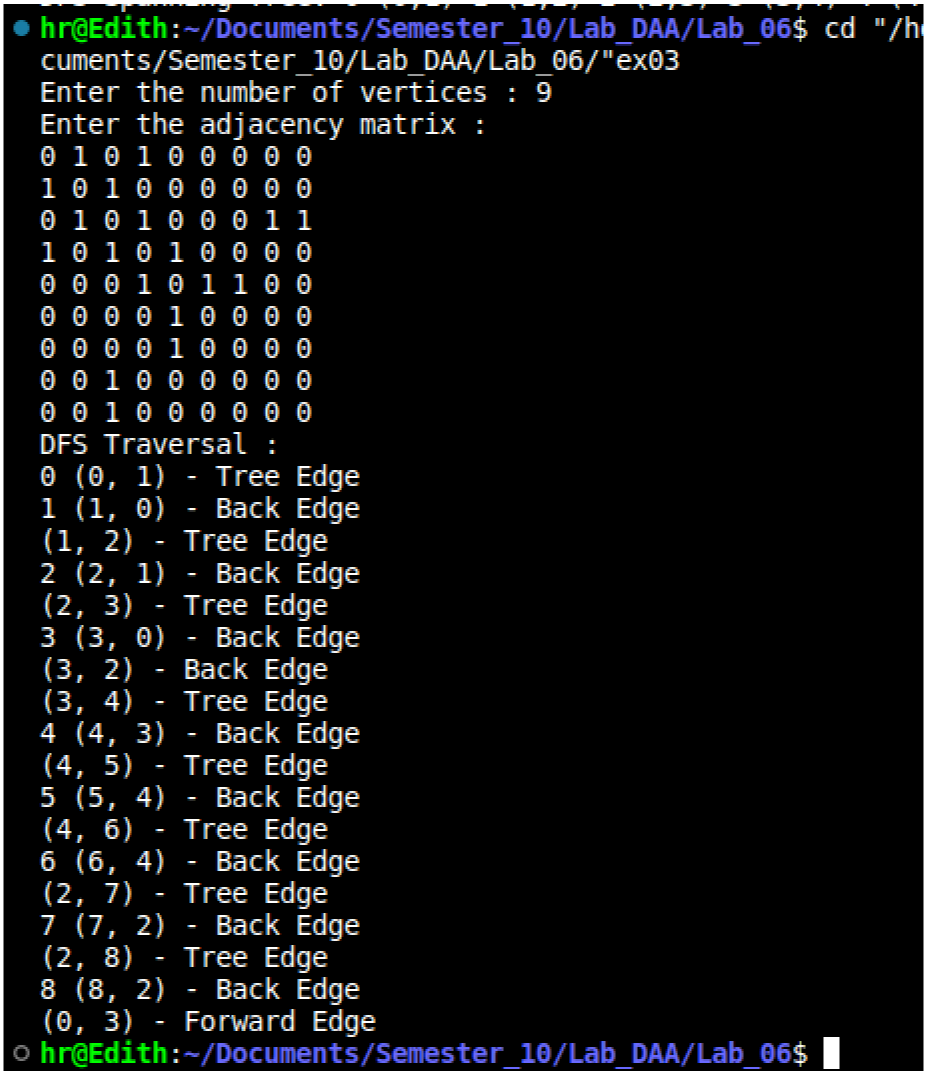
- Code :

```
// DFS
// Tree edge : is an edge that is included in the DFS tree.
// Back edge : An edge from a vertex 'u' to one of its ancestors 'v' is
called as a back edge. A self-loop is considered as a back edge.
// Forward edge : An edge from a vertex 'u' to one of its descendants 'v'
is called as a forward edge.
// Cross edge : An edge from a vertex 'u' to a vertex 'v' that is neither
its ancestor nor its descendant is called as a cross edge.

#include <stdio.h>

#define MAX_VERTICES 100

int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];

void dfs(int vertex, int parent, int n) {
   visited[vertex] = 1;
   printf("%d ", vertex);

   for (int i = 0; i < n; i++) {
       if (graph[vertex][i]) {
           if (!visited[i]) {
               printf("(%d, %d) - Tree Edge\n", vertex, i);
               dfs(i, vertex, n);
           } else if (visited[i] == 1) {
               printf("(%d, %d) - Back Edge\n", vertex, i);
           } else if (visited[i] == 2) {
               printf("(%d, %d) - Forward Edge\n", vertex, i);
           } else {
               printf("(%d, %d) - Cross Edge\n", vertex, i);
           }
       }
   }
   visited[vertex] = 2;
```



```
}

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

   // Initialize graph and visited array
   for (int i = 0; i < n; i++) {
       visited[i] = 0;
       for (int j = 0; j < n; j++) {
           graph[i][j] = 0;
       }
   }

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

   printf("DFS Traversal :\n");
   for (int i = 0; i < n; i++) {
       if (!visited[i]) {
           dfs(i, -1, n);
       }
   }

   return 0;
}
```

- Output Screen-Shots :



```
● hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_06$ cd "/h
  cuments/Semester_10/Lab_DAA/Lab_06/"ex03
  Enter the number of vertices : 9
  Enter the adjacency matrix :
  0 1 0 1 0 0 0 0 0
  1 0 1 0 0 0 0 0 0
  0 1 0 1 0 0 0 1 1
  1 0 1 0 1 0 0 0 0
  0 0 0 1 0 1 1 0 0
  0 0 0 0 1 0 0 0 0
  0 0 0 0 1 0 0 0 0
  0 0 1 0 0 0 0 0 0
  0 0 1 0 0 0 0 0 0
  DFS Traversal :
  0 (0, 1) - Tree Edge
  1 (1, 0) - Back Edge
  (1, 2) - Tree Edge
  2 (2, 1) - Back Edge
  (2, 3) - Tree Edge
  3 (3, 0) - Back Edge
  (3, 2) - Back Edge
  (3, 4) - Tree Edge
  4 (4, 3) - Back Edge
  (4, 5) - Tree Edge
  5 (5, 4) - Back Edge
  (4, 6) - Tree Edge
  6 (6, 4) - Back Edge
  (2, 7) - Tree Edge
  7 (7, 2) - Back Edge
  (2, 8) - Tree Edge
  8 (8, 2) - Back Edge
  (0, 3) - Forward Edge
○ hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_06$
```