TOPOLOGICAL SORT

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
#include <stdlib.h>
#define MAX 100

int graph[MAX][MAX];
char color[MAX];
int parent[MAX];
int visited[MAX];
int d[MAX], finish[MAX];
int max = INT_MAX;
int times = 0;

struct stack
{
   int size;
   int top;
   int *arr;
};

int isEmpty(struct stack *ptr)
{
   if (ptr->top == -1)
   {
      return 1;
   }
   else
   {
      return 0;
```

```
    }
}

int isFull(struct stack *ptr)
{
    if (ptr->top == ptr->size)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void Push(struct stack *ptr, int value)
{
    if (isFull(ptr))
    {
        printf("Stack Overflow! Cannot push %d to the stack\n", value);
    }
    else
    {
        ptr->top++;
        ptr->arr[ptr->top] = value;
        printf("\n%d is pushed into the stack\n",ptr->arr[ptr->top]);
    }
}

int Pop(struct stack *ptr)
{
```

```
    if (isEmpty(ptr))
    {
        printf("Stack Underflow! Cannot pop from the stack\n");
        return -1;
    }
    else
    {
        int value = ptr->arr[ptr->top];
        ptr->top--;
        return value;
    }
}

void instack(struct stack **s)
{
    while (!isEmpty(*s))
    {
        printf("%d ", Pop(*s));
    }
}

void DFS(int graph[MAX][MAX], bool visited[MAX], int vertices, int start, struct stack *s)
{
    printf("%d ", start);
    visited[start] = true;
    color[start] = 'G';
    d[start] = times++;
    for (int i = 0; i < vertices; i++)
    {
        if (graph[start][i] == 1 && !visited[i])
        {
```

```
            parent[i] = start;
            DFS(graph, visited, vertices, i, s);
        }
    }
    color[start] = 'B';
    finish[start] = times++;
    Push(s, start);
}

int main()
{
    int vertices, start;
    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);
    int graph[MAX][MAX];
    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < vertices; i++)
    {
        for (int j = 0; j < vertices; j++)
        {
            scanf("%d", &graph[i][j]);
        }
    }
    for (int i = 0; i < vertices; i++)
    {
        color[i] = 'W';
        parent[i] = -1;
    }
    bool visited[MAX] = {false};
    printf("Enter the starting vertex: ");
    scanf("%d", &start);
```

```
    printf("Depth First Traversal starting from vertex %d: ", start);
    struct stack *sp = (struct stack *)malloc(sizeof(struct stack));
    sp->top = -1;
    sp->size = vertices + 1;
    sp->arr=(int *)malloc(sp->size * sizeof(int));
    DFS(graph, visited, vertices, start, sp);
    printf("\nColor array is: ");
    for (int i = 0; i < vertices; ++i)
    {
        printf("%c ", color[i]);
    }
    printf("\n");
    for (int i = 0; i < vertices; i++)
    {
        printf("Vertex:%d  ", i + 1);
        printf("Weight:%d  ", d[i] + 1);
        printf("final time:%d  ", finish[i] + 1);
        printf("Parent:%d  ", parent[i]);
        printf("\n");
    }
    printf("Stack content: ");
    instack(&sp);
    return 0;
}
```

OUTPUT

```
Enter the number of vertices: 4
Enter the adjacency matrix:
0
1
0
0
0
0
1
0
0
0
0
1
0
0
0
0
Enter the starting vertex: 0
Depth First Traversal starting from vertex 0: 0 1 2 3
3 is pushed into the stack

2 is pushed into the stack

1 is pushed into the stack

0 is pushed into the stack

Color array is: B B B B
Vertex:1  Weight:1  final time:8  Parent:-1
Vertex:2  Weight:2  final time:7  Parent:0
Vertex:3  Weight:3  final time:6  Parent:1
Vertex:4  Weight:4  final time:5  Parent:2
Stack content: 0 1 2 3
```