```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node* next;
};
int push(struct node** head,int element);
int pop(struct node** head,int* element);
int DFS(int source,int total_node,int **adj);
//main program
int main()
{
    //input no of nodes: Num
    int Num;
    printf("Enter number of nodes\n");
    scanf("%d",&Num);
    //create adjacency matrix
    int **adj;
    adj = (int **)malloc((Num) * sizeof(int *));
    for(int i=0; i<Num; i++)
        adj[i] = (int *)malloc((Num) * sizeof(int));
    //input adjacency matrix
    printf("Enter adjacency matrix\n");
    for(int i=0;i<Num;i++)
    {
        for(int j=0;j<Num;j++)
        {
            scanf("%d",&adj[i][j]);
        }
    }
    //DFS traversing
    DFS(0,Num,adj);
    return 0;

}
// dfs function
int DFS(int source,int total_node,int **adj){
    struct node *Stack=NULL;
    int visited[total_node];
    for(int i=0;i<total_node;i++)
    {
        visited[i]=0;
    }
    //add first node to stack
    if(push(&Stack,source)){
        printf("Malloc failed!!!\n");
        return 0;
    }
    int node=-1;
    printf("DFS traversing\n");
    while(Stack!=NULL)
    {
        //get a node from stack and print it
        if(pop(&Stack,&node))
            printf("Stack is empty\n");
        if (!visited[node]){
            visited[node]=1;
            printf("%d ",node);
        }
        //add it's unvisited neighbours to stack
        for(int i=0;i<total_node;i++)
        {
            if(adj[node][i]==1 && visited[i]==0)
                if(push(&Stack,i)){
                    printf("Malloc failed!!!\n");
                    return 0;
                }
        }
    }
    printf("\n");
    return 0;
}
//stack functions
int push(struct node** head,int element)
{
    struct node* temp;
    temp=*head;
    *head=(struct node*)malloc(sizeof(struct node));
    if(head==NULL)
        return 1;
    (*head)->data=element;
    (*head)->next=temp;
    return 0;
}
int pop(struct node** head,int* element)
{
    if(*head==NULL)
        return 1;
    *element=(*head)->data;
    struct node* temp;
    temp=*head;
    *head=(*head)->next;
    free(temp);
    return 0;
}
```