ASSIGNMENT NO:4 **DATE:01/03/2016**

PROGRAM TITLE:Traverse a Graph which is represented by Adjacency List using Depth First Search.

PROGRAM ALGORITHM:

```
DFS(G,v)
{
      //input:Graph 'G' represented either by adjacency matrix, starting vertex
'v'
      //output:printing the vertices in DFS order
      push(S,v);//S is an empty Queue
      while S is not empty do
      {
            r=pop(S);
            if mark[r]==FALSE
            {
                  mark[u]=TRUE;
                  print(r);
                  for each adjacent vertex 'u' of 'r' do
                  {
                        push(S,u);
                  }
            }
      }
}
main()
{
      print(starting vertex);
      for each vertex u
            mark[u]=FALSE;
      DFS(G,v);
      for all unmarked vertices
            DFS(G,w);
}
```

PROGRAM CODE:

```
//C Code to Traverse a given graph using DFS using Adjacency List
#include <stdio.h>
#include <stdlib.h>
#define MAX 50
int stack[MAX];
int top=-1;
struct Node
{
      int data;
      struct Node *next;
};
typedef struct Node *NODEPTR;
NODEPTR allocate_node(int item)//Allocates memory space for a new node
{
      NODEPTR temp = (NODEPTR)malloc(sizeof(struct Node));
      temp->data=item;
      temp->next=NULL;
      return temp;
}
void create_graph(struct Node **graph,int n)
```

```
{
      int i=0,j;
      for(i=0;i<n;i++)
      {
            NODEPTR start=NULL;
            int item;
            printf("\n\tEnter the adjoining vertices of %d. Enter -99 to
stop.",i);
            for(j=0;j<n;j++)
            {
                  scanf("%d",&item);
                  if(item==-99)
                        break;
                  else
                  {
                        NODEPTR temp=allocate_node(item);
                        if(start==NULL)
                        {
                              start=temp;
                        }
                        else
                        {
                              NODEPTR p=start,q;
                              while((p!=NULL)&&((temp->data)>(p->data)))
                              {
                                    q=p;
                                    p=p->next;
                              }
                              temp->next=p;
                              if(p==start)
                                    start=temp;
                              else
                                    q->next=temp;
                        }
                  }
            }
            (graph[i])->next=start;
      }
}
void print_graph(struct Node **graph,int n)
{
      int i;
      printf("\n\tThe Adjacency List of the Graph is::");
      for(i=0;i<n;i++)
      {
            printf("\n");
            printf("Vertex%d",(graph[i])->data);
            if((graph[i])->next==NULL)
            {
                  printf("\n\tThe vertice has no adjacent vertices.");
            }
            else
            {
                  NODEPTR p=(graph[i])->next;
                  while(p!=NULL)
                  {
                        printf("\t->%d",p->data);
```

```
                        p=p->next;
                }
            }
        }
        printf("\n");
}
int is_full()
{
        if(top==MAX-1)
            return 1;
        else
            return 0;
}
int is_empty()
{
        if(top==-1)
            return 1;
        else
            return 0;
}
void push(int n)
{
        if(is_full()==0)
        {
            top++;
            stack[top]=n;
        }
        else
            printf("\tThe Stack is Full.");
}
int pop()
{
        if(is_empty()==0)
        {
            return(stack[top--]);
        }
        else
            printf("\tThe Stack is Empty.");
        return -9999;
}
int dfs(struct Node **graph,int n,int *mark,int v)
{
        int r;
        top=-1;
        push(v);
        while(is_empty()==0)
        {
            r=pop();
            if(r!=-9999)
            {
                if(mark[r]==0)
                {
                    mark[r]=1;
                    printf("\t%d",r);
                    NODEPTR p=(graph[r])->next;
                    while(p!=NULL)
                    {
```

```
                        push(p->data);
                        p=p->next;
                    }
                }
            }
        }
        return 0;
}
int main()
{
        int n,i,v;
        printf("\n\tEnter the number of vertices::");
        scanf("%d",&n);
        int *mark=(int*)malloc(n*sizeof(int));
        for(i=0;i<n;i++)
            mark[i]=0;
        struct Node **graph= (struct Node**)malloc(n*sizeof(struct Node*));
        for(i=0;i<n;i++)
        {
            graph[i]=(struct Node*)malloc(sizeof(struct Node));
            graph[i]->data=i;
            graph[i]->next=NULL;
        }
        create_graph(graph,n);
        print_graph(graph,n);
        do
        {
            printf("\n\tEnter the starting vertex. Enter -99 to Quit::");
            scanf("%d",&v);
            for(i=0;i<n;i++)
                mark[i]=0;
            if(v!=-99)
            {
                printf("\tThe DFS traversal is::\n");
                dfs(graph,n,mark,v);
                printf("\n");
                for(i=0;i<n;i++)
                {
                    if(mark[i]==0)
                    {
                        dfs(graph,n,mark,i);
                        printf("\n");
                    }
                }
            }
        }
        while(v!=-99);
        return 0;
}
```

OUTPUT:

```
        Enter the number of vertices::5

        Enter the adjoining vertices of 0. Enter -99 to stop.1 2 3 -99

        Enter the adjoining vertices of 1. Enter -99 to stop.0 4 -99
```

```
     Enter the adjoining vertices of 2. Enter -99 to stop.0 4 -99

     Enter the adjoining vertices of 3. Enter -99 to stop.0 4 -99

     Enter the adjoining vertices of 4. Enter -99 to stop.1 2 3 -99

     The Adjacency List of the Graph is::
Vertex0    ->1   ->2   ->3
Vertex1    ->0   ->4
Vertex2    ->0   ->4
Vertex3    ->0   ->4
Vertex4    ->1   ->2   ->3

     Enter the starting vertex. Enter -99 to Quit::0
     The DFS traversal is::
     0     3     4     2     1

     Enter the starting vertex. Enter -99 to Quit::1
     The DFS traversal is::
     1     4     3     0     2

     Enter the starting vertex. Enter -99 to Quit::2
     The DFS traversal is::
     2     4     3     0     1

     Enter the starting vertex. Enter -99 to Quit::3
     The DFS traversal is::
     3     4     2     0     1

     Enter the starting vertex. Enter -99 to Quit::4
     The DFS traversal is::
     4     3     0     2     1

     Enter the starting vertex. Enter -99 to Quit::-99
```

DISCUSSION:

For Depth First Search, principle of Stack is used, therefore, we had to design some more functions to maintain and perform operations on the stack.