//GAYATHRI .M 185001050

// GRAPH

Program:
Contents of functions.h file

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct vertex
{
	char v;
	char adj[10];
	int visit;
}node;
typedef struct stack
{
	node *ver;
	struct stack *next;
}stack;
typedef struct queue
{
	node *ver;
	struct queue *next;
}queue;
int n;
stack * push(node * v,stack * top)
{
	stack *ptr=malloc(sizeof(stack));
	ptr->ver=v;
	ptr->next=top;
	top=ptr;
	return top;
}
```

```
node * pop(stack **top)
{
        if(*top!=NULL)
        {
                node *ptr=(*top)->ver;
                stack *temp =(*top);
                (*top)=(*top)->next;
                temp->next=NULL;
                free(temp);
                return ptr;
        }
        return NULL;
}
void enqueue(node * v,queue **front,queue **rear)
{
        queue *ptr = (queue*)malloc(sizeof(queue));
        ptr->ver=v;
        ptr->next=NULL;
        if(*front==NULL)
                *front=*rear=ptr;
        else
        {
                (*rear)->next=ptr;
                *rear=ptr;
        }
}
node * dequeue(queue **front,queue **rear)
{
        queue *temp=(*front);
        node * ptr=(*front)->ver;
        (*front)=(*front)->next;
        free(temp);
        return ptr;
}
void dfs(node *v,stack *top,node *gp[10])
{
        if(v== NULL)
                return;
        printf("\nDepth first search \n");
        top=push(v,top);
        while(top!=NULL)
        {
                node * temp=pop(&top);
                if(temp->visit==0)
                {
                    temp->visit=1;
                    printf("%c\t",temp->v);
                }
                for(int i=0;temp->adj[i];++i)
                {
                    for(int j=0;j<n;++j)
                    {
                        if((temp->adj[i]==gp[j]->v)&&gp[j]->visit==0)
                            top=push(gp[j],top);
                    }
                }
        }
}
node * unvisit(node *gp[])
{
        for(int i=0;i<n;++i)
                if(gp[i]->visit==0)
                    return gp[i];
        return  NULL;
}
void bfs(node *v,node *gp[10])
{
        queue *front,*rear;
        front=NULL;
        rear=NULL;
        printf("\n\nBreadth first search \n");
        while(unvisit(gp)!=NULL)
        {
                node * vnode=unvisit(gp);
                if(vnode){
                printf("%c\t",vnode->v);
                vnode->visit=1;
                enqueue(vnode,&front,&rear);
                }
                while(front!=NULL)
                {
                    node *temp=dequeue(&front,&rear);
                    for(int i=0;temp->adj[i];++i)
                    {
                        for(int j=0;j<n;++j)
                        {
                            if((temp->adj[i]==gp[j]->v)&&gp[j]->visit==0)
                            {
                                printf("%c\t",gp[j]->v);
                                gp[j]->visit=1;
                                enqueue(gp[j],&front,&rear);

                            }
                        }
                    }
                }
        }

}
node * create()
{
        char c;
        int i=0;
        node *newv=malloc(sizeof(node));
        printf("\nEnter vertex: ");
        scanf(" %c",&newv->v);
        newv->visit=0;
        printf("\nEnter Adjacency list of %c (*-stop): \n",newv->v);
        scanf(" %c",&c);
        while(c!='*')
        {
```

```
                newv->adj[i]=c;
                i+=1;
                scanf(" %c",&c);
        }
        newv->adj[i]='\0';
        return newv;
}
char *Strrev(char *str)
{
     char *p1, *p2;

     if (! str || ! *str)
          return str;
     for (p1 = str, p2 = str + strlen(str) - 1; p2 > p1; ++p1, --p2)
     {
          *p1 ^= *p2;
          *p2 ^= *p1;
          *p1 ^= *p2;
     }
     return str;
}
```

Contents of graph.c file

```
#include"functions.h"
int main()
{
   int op;
   do{
        printf("\nEnter no of Vertices: ");
        scanf("%d",&n);
        node * v[10];
        stack *top=NULL;
        for(int i=0;i<n;++i)
        {       v[i]=create();
                Strrev(v[i]->adj);}
        dfs(v[0],top,v);
        for(int i=0;i<n;++i)
        {       v[i]->visit=0;
                Strrev(v[i]->adj);
                }

        bfs(v[0],v);
        printf("\n\nDo you want to continue(1-yes/0-no): ");
        scanf("%d",&op);
   }while(op==1);
        return 0;
}
```

Output:

```
Enter no of Vertices: 5

Enter vertex: A

Enter Adjacency list of A (*-stop):
B
C
E
*

Enter vertex: B
Enter Adjacency list of B (*-stop):
A
D
E
*

Enter vertex: C
Enter Adjacency list of C (*-stop):
A
*

Enter vertex: D

Enter Adjacency list of D (*-stop):
B
*

Enter vertex: E

Enter Adjacency list of E (*-stop):
A
B
*

Depth first search
A       B       D       E       C

Breadth first search
A       B       C       E       D

Do you want to continue(1-yes/0-no): 1

Enter no of Vertices: 5

Enter vertex: 0

Enter Adjacency list of 0 (*-stop):
1
*

Enter vertex: 1
Enter Adjacency list of 1 (*-stop):
2
*
```

```
Enter vertex: 2

Enter Adjacency list of 2 (*-stop):
3
4
*

Enter vertex: 3

Enter Adjacency list of 3 (*-stop):
0
*

Enter vertex: 4

Enter Adjacency list of 4 (*-stop):
2
*

Depth first search
0       1       2       3       4

Breadth first search
0       1       2       3       4

Do you want to continue(1-yes/0-no): 0
```