ASSIGNMENT NO:3 **DATE:23/02/2016**

PROGRAM TITLE:Traverse a Graph using Breadth First Search using Adjacency List.

PROGRAM ALGORITHM:

```
BFS(G,v)
{
    //input:Graph 'G' represented either by adjacency matrix or adjacency list, starting vertex 'v'
    //output:printing the vertices in BFS order
    enqueue(Q,v);//Q is an empty Queue
    mark[v]=TRUE;
    while Q is not empty do
    {
        r=dequeue(Q);
        print(r);
        for each adjacent vertex 'u' of 'r' do
        {
            if mark[u]==FALSE
            {
                mark[u]=TRUE;
                enqueue(Q,u);
            }
        }
    }
}

main()
{
    print(starting vertex);
    for each vertex u
        mark[u]=FALSE;
    BFS(G,v);
    for all unmarked vertices
        BFS(G,w);
}
```

PROGRAM CODE:

```
//C Code to Traverse a given graph using BFS using Adjacency List
#include <stdio.h>
#include <stdlib.h>
#define MAX 50
int q[MAX];
int front=0,rear=0;
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,j;
    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((rear+1)%MAX==front)
                return 1;
          else
                return 0;
    }
    int is_empty()
    {
          if(front==rear)
                return 1;
          else
                return 0;
    }
    void enqueue(int n)
    {
          if(is_full()==0)
          {
                q[rear]=n;
                rear=(rear+1)%MAX;
          }
          else
                printf("\tThe Queue is Full.");
    }
    int dequeue()
    {
          if(is_empty()==0)
          {
                int x=q[front];
                front=(front+1)%MAX;
                return x;
          }
          else
                printf("\tThe Queue is Empty.");
          return -9999;
    }
    int bfs(struct Node **graph,int n,int *mark,int v)
    {
          int r,i;
          front=rear=0;
          enqueue(v);
          mark[v]=1;
          enqueue(-99);
          while(is_empty()==0)
          {
                r=dequeue();
```

```
        if(r!=-9999)
        {
            if(r==-99)
                printf("\n");
            else
            {
                printf("\t%d",r);
                NODEPTR p=(graph[r])->next;
                while(p!=NULL)
                {
                    if(mark[p->data]==0)
                    {
                        mark[p->data]=1;
                        enqueue(p->data);
                    }
                    p=p->next;
                }
                enqueue(-99);
            }
        }
    }
    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 BFS traversal is::\n");
            bfs(graph,n,mark,v);
            for(i=0;i<n;i++)
            {
                if(mark[i]==0)
                    bfs(graph,n,mark,i);
            }
        }
    }
```

```
        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 BFS traversal is::
        0
        1     2     3
        4




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




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




        Enter the starting vertex. Enter -99 to Quit::3
        The BFS traversal is::
        3
        0     4
```

```
1     2




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




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

DISCUSSION:

1. For Breadth First Search, principle of Queue is used, therefore, we had to design some more functions to maintain and perform operations on the queue.
2. We create an Adjacency List for storing the graph in.