ASSIGNMENT NO 6

Represent a given graph using adjacency list to perform DFS and BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and perform DFS and BFS on that.

```
//**************DFS AND BFS TRAVERSAL*************
#include<iostream>
#include<string>
#define MAX_SIZE 100
using namespace std;
 class Queue
 {
public:
     int front,rear;
     int myqueue[MAX_SIZE];
     Queue()
     {
         front=-1;
         rear=-1;
     }
     void enQueue(int v)
     {
         if(isFull())
         {
             cout<<"Queue is full";
         }
         else
         {
             if(front==-1)
             {
                 front=0;
             }
             rear++;
            myqueue[rear]=v;
         }
     }
     bool isFull()
     {
         if(front==0 && rear==MAX_SIZE-1)
         {
             return true;
         }
         else
         {
             return false;
         }
```

```
    }
    bool isEmpty()
    {
        if(front==-1)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    int delQueue()
    {
        int value;
        if(isEmpty())
        {
            cout<<"Queue is empty";
            return -1;
        }
        else
        {
            value=myqueue[front];
            if(front>=rear)
            {
                front=-1;
                rear=-1;
            }
            else
            {
                front++;
            }
        }
        return value;
    }
};

class Node
{
    int des;
    Node *next;
    friend class Graph;
public:
    Node(int d=0)
    {
```

```
        des=d;
        next=NULL;
    }
};

class Graph
{
    int v;
    int e;
    Node **header;
    bool *visit;
    string *name;
public:
    Graph(int vert=0,int edge=0)
    {
        v=vert;
        e=edge;
        header=new Node*[v];
        visit=new bool[v];
        name=new string[v];
        for(int i=0;i<v;i++)
        {
            header[i]=NULL;
            visit[i]=0;
            name[i]="";
        }
    }
    void reset_visit()
    {
        for(int i=0;i<v;i++)
        {
            visit[i]=0;
        }
    }
    int search(string key)
    {
        for(int i=0;i<v;i++)
        {
            if(name[i]==key)
            {
                return i;
            }
        }
    }
    void accept()
```

```
    {
        cout<<"Enter the name of all locations:";
        for(int i=0;i<v;i++)
        {
            cout<<"\nLocation:"<<i+1<<" : ";
            cin>>name[i];
        }
        for(int i=0;i<v;i++)
        {
            int src,dest;
            string pos1,pos2;
            cout<<"\nEnter Location 1:";
            cin>>pos1;
            cout<<"\nEnter Location 2:";
            cin>>pos2;
            src=search(pos1);
            dest=search(pos2);
            cout<<pos1<<"    "<<src<<"   ";
            cout<<pos2<<"    "<<dest<<"   ";
            Node *p=new Node(dest);
            Node *q=new Node(src);
            if(header[src]==NULL)
            {
                header[src]=p;
            }
            else
            {
                Node *temp=header[src];
                while(temp->next!=NULL)
                {
                    temp=temp->next;
                }
                temp->next=p;
            }
            if(header[dest]==NULL)
            {
                header[dest]=q;
            }
            else
            {
                Node *temp=header[dest];
                while(temp->next!=NULL)
                {
                    temp=temp->next;
                }
```

```
                temp->next=q;
            }
        }
    }

    void DFS(int v)
    {
        cout<<name[v]<<" ";
        visit[v]=1;
        Node *temp=header[v];
        while(temp!=NULL)
        {
            int t=temp->des;
            if(visit[t]==false)
            {
                DFS(t);
            }
            temp=temp->next;
        }
    }
    void BFS(int v)
    {
        Queue q;
        visit[v]=1;
        q.enQueue(v);
        cout<<name[v]<<" ";
        while(!q.isEmpty())
        {
            Node *temp=header[v];
            if(visit[v]==false)
            {
                cout<<"Name "<<name[v]<<" ";
                visit[v]=1;
            }
            while(temp!=NULL)
            {
                if(visit[temp->des]==false)
                {
                    q.enQueue(temp->des);
                    visit[temp->des]=1;
                    cout<<name[temp->des]<<" ";
                }
                temp=temp->next;
            }
            q.delQueue();
```

```
            v++;
        }
    }
};

int main()
{
int a, b, s;
int ch;

cout << "Enter Number of Vertices: ";
cin >> a;
cout << "Enter Number of Edges: ";
cin >> b;
Graph g1(a, b);
g1.accept();
do
    {
        cout<<"\n**********MENU*********";
        cout<<"\n1.DFS";
        cout<<"\n2.BFS";
        cout<<"\n3.Exit";
        cout<<"\nEnter your choice:";
        cin>>ch;
        switch(ch)
        {
        case 1:
        cout << "\nDFS : ";
        cout << "\nEnter Start Index:";
        cin >> s;
        g1.DFS(s);
        break;

        case 2:
        g1.reset_visit();
        cout << "\nBFS : ";
        cout << "\nEnter Start Index:";
        cin >> s;
        g1.BFS(s);
        break;
        case 3:
            cout<<"Thank you for using this program!";
            exit(0);
            break;
        default:
```

```
            cout<<"Enter correct choice"<<endl;
            break;
        }
    }while(ch!=3);
    return 0;
}
```