12. Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform 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.

Code:

```
#include <iostream>
using namespace std;
#define max 10

class graph
{
        int stack[max];
        int top;

        int queue[max];
        int front, rear;

        struct node
        {
                int data;
                node *next;

        }*head[max];

        int n;
        public:
                graph()
                {
                        cout<<"\nEnter the no. of nodes: ";
                        cin>>n;
                        for (int i=0; i<n; i++)
                           head[i]=NULL;

                        top=-1;

                        front=-1;
                        rear=-1;
                }

        public:
                void create();
                void dfs();
                void push(int num);
                int pop();
                int empty();

                void bfs();
                void insert(int val);
                int Qempty();
                int del();
};

void graph::create()
{
        node *temp, *curr;
        //char flag;
        int anode;
        int i=0;

  do
        {
        temp=new node;
        temp->next=NULL;
        head[i]=temp;
        curr=head[i];
        cout<<"Enter the value of vertex: "<<endl;
        cin>>temp->data;

        cout<<"\nHow many adjecent nodes does the above vertex have: ";
        cin>>anode;

        for (int k=0; k<anode; k++)
        {
                temp=new node;
                temp->next=NULL;
                cout<<"\nEnter the Adjecent Vertex"<<endl;
                cin>>temp->data;
                curr->next=temp;
                curr=temp;
        }
        /*cout<<"\nWant to add a new head node?(y/n)";
        cin>>flag;*/
  i++;
  }while(i<n);
```

```
}

void graph::push(int num)
{
	if(top==max-1)
	{
		cout<<"\nYou crossed the max limit of array!!";
		return;
	}
	else
	{
		top=top+1;
		stack[top]=num;

	}
}

int graph::empty()
{
	if(top==-1)
	{
		return true;
	}
	else
		return false;
}

int graph::pop()
{
	if (empty()==true)
	{
		return 0;
	}
	else
	{
		int num = stack[top];
		top = top - 1;
		return num;
	}
}

void graph::dfs()
{
	node *temp;
```

```
        int visited[n], num, sv;
        for(int i=0;i<n; i++)
        {
                visited[i]=0;
        }

        cout<<"\nEnter the Starting Vertex: ";
        cin>>sv;
        push(sv);

        while(empty()!=true)
        {
                num=pop();
                if(visited[num]==0)
                {
                        cout<<num<<" ";
                        visited[num]=1;
                }

                temp=head[num];

                while(temp!=NULL)
                {
                        if(visited[temp->data]==0)
                        {
                                push(temp->data);
                        }
                        temp=temp->next;
                }
        }
        cout<<endl;
}

void graph::insert(int val)
{
  if (rear == max - 1)
  {
     return;
  }
  else
  {

                if (front == - 1)
    {
            front = 0;
                rear = 0;
           }
                else
                rear++;
      queue[rear] = val;
  }
}

int graph::Qempty()
{
        if(front == -1 || front > rear)
        {
                return true;
        }
        else
                return false;
}
int graph::del()
{
  if (Qempty()==true)
  {
    return 0;
  }
  else
  {
    int num = queue[front];
    front=front+1;
    return num;
  }
}

void graph::bfs()
{
        node *temp;
        int visited[n], num, sv;
        for(int i=0;i<n; i++)
        {
                visited[i]=0;
        }

        cout<<"\nEnter the Starting Vertex: ";
        cin>>sv;
        insert(sv);
        while(Qempty()!=true)
        {
                num=del();
                if(visited[num]==0)
                {
                    cout<<num<<" ";
                    visited[num]=1;
                }

                temp=head[num];

                while(temp!=NULL)
                {
                    if(visited[temp->data]==0)
                    {
                        insert(temp->data);
                    }
                    temp=temp->next;
                }
        }

}

int main()
{
        graph obj;
        char ans;
        do
        {
        int flag;
      cout<<"\n What Operation do you want to perform?";
        cout<<"\n 1.)Create Graph \n 2.)DFS \n 3.)BFS \n 4.)Exit \n";
        cin>>flag;
        if(flag==1)
        {
                obj.create();
        }
        else if(flag==2)
        {
                obj.dfs();
        }
        else if(flag==3)
        {
```

```
                obj.bfs();
        }
        else if(flag==4)
        {
                cout<<"\n **** Thank You! **** \n";
        }
        else
        {
                cout<<"Please re-check your choice.";
        }
        cout<<"\n Do you want to Continue?";
        cout<<"\n Yes=y \n No=n \n";
        cin>>ans;
   }while(ans=='y');
        return 0;
}
```