Practical-6(C-13)

Problem Statement:

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 SIZE 10
#define MAX 20

class Queue
{
   int front,rear;
   int arr[MAX];

   public:
   Queue()
   {
      front=rear=-1;
   }
   void insert_ele(int x)
   {
      if(!isFull())
      {
         if(front==-1)
            front++;
         rear++;
         arr[rear]=x;
      }
   }
   int delete_ele()
   {
      if(!isEmpty())
      {
         int x=arr[front];
         for(int i=front;i<rear;i++)
         {
            arr[i]=arr[i+1];
         }
         if(front==rear)
            front--;
         rear--;
         return x;
      }
   }
   bool isEmpty()
   {
      if(rear==-1)
         return true;
      else
         return false;
   }
   bool isFull()
   {
      if(rear==MAX-1)
         return true;
      else
         return false;
   }
   void display_queue()
   {
      if(!isEmpty())
      {
            cout<<"\nQueue : ";
            for(int i=front;i<=rear;i++)
                cout<<arr[i]<<" ";
        }

    }
};

class Stack
{
    int arr[MAX];
    int top;
    public:
    Stack()
    {
        top=-1;
    }
    void push(int x)
    {
        if(!isFull())
        {
            top++;
            arr[top]=x;
        }
    }
    int pop()
    {
        if(!isEmpty())
        {
            int x=arr[top];
            top--;
            return x;
        }
    }
    bool isEmpty()
    {
        if(top==-1)
            return true;
        else
            return false;
    }
    bool isFull()
    {
        if(top==MAX-1)
            return true;
        else
            return false;
    }
    void display_stack()
    {
        cout<<"\nStack : ";
        for(int i=top;i>=0;i--)
            cout<<arr[i]<<" ";
    }
};

class Graph
{

    int i,j;
    int visited_arr[20];

    public:

    int cnt;
    int adj_mat[SIZE][SIZE];
    int ver_arr[SIZE];
    int vertex_count;

    Graph()
    {
        cnt=0;
    }

    void DFS();
    void BFS();

    void display(string s)
    {
        cout<<"\n"<<s<<" Traversal : ";
        for(i=0;i<cnt;i++)
        {
            cout<<visited_arr[i]<<" ";
        }
    }

    int search(int x)
    {
        for(i=0;i<cnt;i++)
        {
            if(visited_arr[i]==x)
                return 1;
        }
        return 0;
    }

    void create_adjmat()
    {
        cout<<"\nEnter the total number of nodes in the graph (less than 10) : ";
        cin>>vertex_count;
        cout<<"\nEnter the values of the nodes :\n";
        for(i=1;i<=vertex_count;i++)
        {
            cin>>ver_arr[i];
        }
        cout<<"\nEnter 1 if edge present between nodes else enter 0\n";
        for(i=1;i<=vertex_count;i++)
        {
            for(j=1;j<=vertex_count;j++)
            {
                cout<<"Node "<<ver_arr[i]<<" and Node "<<ver_arr[j]<<" : ";
                cin>>adj_mat[i][j];
            }
        }
    }
    void display_adjmat()
    {
        cout<<"\nAdjacency Matrix \n\n  ";
        for(i=1;i<=vertex_count;i++)
            cout<<ver_arr[i]<<" ";
        cout<<"\n";
        for(i=1;i<=vertex_count;i++)
        {
            cout<<ver_arr[i]<<" ";
            for(j=1;j<=vertex_count;j++)
            {
                cout<<adj_mat[i][j]<<" ";
            }
            cout<<"\n";
        }
    }
};
```

```
void Graph :: DFS()
{
   int curr_ver;
   Stack sobj;
   int i,j,flag=0;
   cout<<"Enter starting vertex : ";
   cin>>curr_ver;
   cnt=0;
   sobj.push(curr_ver);
   visited_arr[cnt++]=curr_ver;
   curr_ver = sobj.pop();
   for(i=curr_ver;i<=vertex_count;)
   {
      if(flag ==0)
      {
         for(j=1;j<=vertex_count;j++)
         {

            if(adj_mat[i][j]==1)
            {
               sobj.push(j);
            }
         }
         flag=1;
      }
      curr_ver = sobj.pop();

      if(search(curr_ver)==0)
      {
         visited_arr[cnt++]=curr_ver;
         flag=0;
      }

      i=curr_ver;
      if(cnt == vertex_count)
      {
         break;
      }

   }

   display("DFS");
}

void Graph :: BFS()
{
   int curr_ver;
   Queue qobj;
   int i,j,flag=0;
   cout<<"Enter starting vertex : ";
   cin>>curr_ver;
   cnt=0;
   qobj.insert_ele(curr_ver);
   visited_arr[cnt++]=curr_ver;
   curr_ver = qobj.delete_ele();
   for(i=curr_ver;i<=vertex_count;)
   {
      if(flag ==0)
      {
         for(j=1;j<=vertex_count;j++)
         {

            if(adj_mat[i][j]==1)
            {
               qobj.insert_ele(j);
```

```
                }
            }
            flag=1;
        }
        curr_ver = qobj.delete_ele();

        if(search(curr_ver)==0)
        {
            visited_arr[cnt++]=curr_ver;
            flag=0;
        }

        i=curr_ver;
        if(cnt == vertex_count)
        {
            break;
        }

    }

    display("BFS");
}

int main()
{
    Graph gobj;
    int choice;

    do
    {
        cout<<"\n\t\t\tMENU";
        cout<<"\n\t1. Create Matrix\n\t2. Display Matrix\n\t3. Perform DFS Traversal\n\t4. Perform BFS Traversal\n\t5. Exit\n";
        cout<<"\n Enter your choice : ";
        cin>>choice;

        switch(choice)
        {
            case 1: gobj.create_adjmat();
                    break;
            case 2: gobj.display_adjmat();
                    break;
            case 3: gobj.DFS();
                    break;
            case 4: gobj.BFS();
                    break;
            case 5: cout<<"Exited Code";
                     break;
            default: cout<<"Invalid Option Chosen!";
        }
    }while(choice!=5);
    return 0;
}
```

OUTPUT:

```
MENU
              1. Create Matrix
              2. Display Matrix
              3. Perform DFS Traversal
             4. Perform BFS Traversal
              5. Exit

 Enter your choice : 1
 Enter the total number of nodes in the graph (less than 10) : 3
 Enter the values of the nodes :
```

```
1
2
3
Enter 1 if edge present between nodes else enter 0
Node 1 and Node 1 : 0
Node 1 and Node 2 : 1
Node 1 and Node 3 : 1
Node 2 and Node 1 : 1
Node 2 and Node 2 : 0
Node 2 and Node 3 : 1
Node 3 and Node 1 : 1
Node 3 and Node 2 : 1
Node 3 and Node 3 : 0
MENU
            1. Create Matrix
            2. Display Matrix
            3. Perform DFS Traversal
            4. Perform BFS Traversal
            5. Exit

 Enter your choice : 2
 Adjacency Matrix

  1 2 3
1 0 1 1
2 1 0 1
3 1 1 0

                                    MENU
            1. Create Matrix
            2. Display Matrix
            3. Perform DFS Traversal
            4. Perform BFS Traversal
            5. Exit

 Enter your choice : 3
 Enter starting vertex : 1
 DFS Traversal : 1 3 2
                                    MENU
            1. Create Matrix
            2. Display Matrix
            3. Perform DFS Traversal
            4. Perform BFS Traversal
            5. Exit

 Enter your choice : 4
 Enter starting vertex : 1
 BFS Traversal : 1 2 3
                                    MENU
            1. Create Matrix
            2. Display Matrix
            3. Perform DFS Traversal
            4. Perform BFS Traversal
            5. Exit

 Enter your choice : 5
```