Code Explanation Assistance

i have viva and practical exam
if i provide you my code then can you help me by explaining the code line by line,also what is use of the code and how it works
and if possible can you give video to understand the concept in code

Yes, absolutely! If you share your code with me, I can:

1. ✅ **Explain it line by line** in clear and simple language.
2. ✅ **Tell you the purpose** of the code — what it's used for and where it's applied.
3. ✅ **Describe how it works** — the logic and flow behind the program.
4. ✅ **(If possible)** — I'll also try to find or recommend a **relevant video** that explains the same concept visually (e.g., on YouTube).

Just upload or paste your code here, and I'll start with the explanations.

What's the subject or topic of the code (like AVL Trees, Hashing, Graphs, etc.)?

```
#include<iostream>
using namespace std;

class tree
{
	int a[20][20],l,u,w,i,j,v,e,visited[20];
public:
	void input();
	void display();
	void minimum();
};

void tree::input()
{
	cout<<"Enter the no. of branches: ";
	cin>>v;

	for(i=0;i<v;i++)
	{
		visited[i]=0;
		for(j=0;j<v;j++)
		{
			a[i][j]=999;
		}
```

```
        }

        cout<<"\nEnter the no. of connections: ";
        cin>>e;

        for(i=0;i<e;i++)
        {
                cout<<"Enter the end branches of connections:  "<<endl;
                cin>>l>>u;
                cout<<"Enter the phone company charges for this connection:  ";
                cin>>w;
                a[l-1][u-1]=a[u-1][l-1]=w;
        }
}

void tree::display()
{
        cout<<"\nAdjacency matrix:";
        for(i=0;i<v;i++)
        {
                cout<<endl;
                for(j=0;j<v;j++)
                {
                        cout<<a[i][j]<<"    ";
                }
                cout<<endl;
        }
}

void tree::minimum()
{
        int p=0,q=0,total=0,min;
        visited[0]=1;
        for(int count=0;count<(v-1);count++)
        {
                min=999;
                for(i=0;i<v;i++)
                {
                        if(visited[i]==1)
                        {
                                for(j=0;j<v;j++)
                                {
                                        if(visited[j]!=1)
                                        {
                                                if(min > a[i][j])
                                                {
                                                        min=a[i][j];
                                                        p=i;
                                                        q=j;
                                                }
                                        }
```

```
                        }
                    }
                }
                visited[p]=1;
                visited[q]=1;
                total=total+min;
                cout<<"Minimum cost connection is"<<(p+1)<<" -> "<<(q+1)<<"  with charge : "<<min<<
endl;

        }
        cout<<"The minimum total cost of connections of all branches is: "<<total<<endl;
}

int main()
{
        int ch;
        tree t;
        do
        {
                cout<<"==========PRIM'S ALGORITHM================="<<endl;
                cout<<"\n1.INPUT\n \n2.DISPLAY\n \n3.MINIMUM\n"<<endl;
                cout<<"Enter your choice :"<<endl;
                cin>>ch;

        switch(ch)
        {
        case 1: cout<<"*******INPUT YOUR VALUES*******"<<endl;
                t.input();
                break;

        case 2: cout<<"*******DISPLAY THE CONTENTS********"<<endl;
                t.display();
                break;

        case 3: cout<<"*********MINIMUM***********"<<endl;
                t.minimum();
                break;
        }

        }while(ch!=4);
        return 0;
}
```

Code Explanation (Line by Line)

This code implements **Prim's Algorithm** for finding the **Minimum Spanning Tree (MST)** of a connected graph, which is often used to minimize the total cost for a network. In this case, it simulates a network of phone company connections with costs between branches.

Header Files & Class Declaration

```cpp
#include<iostream>
using namespace std;

class tree
{
    int a[20][20], l, u, w, i, j, v, e, visited[20];
public:
    void input();
    void display();
    void minimum();
};
```

- `#include<iostream>`: This includes the input-output stream for taking inputs and printing outputs.
- `using namespace std;`: To avoid prefixing `std::` with standard library functions.
- **Class `tree`**: A class is defined with the following variables:
  - `a[20][20]`: Adjacency matrix for storing the graph (with maximum 20 vertices).
  - `l, u, w, i, j, v, e`: Helper variables for branches, edges, costs, and loops.
  - `visited[20]`: To track the visited nodes during the algorithm (Maximum 20 vertices).

Methods in the class:

- `input()`: To take input for the graph (vertices and edges).
- `display()`: To display the adjacency matrix.
- `minimum()`: Implements Prim's algorithm to find the minimum spanning tree.

`input()` Method

```cpp
void tree::input()
{
    cout << "Enter the no. of branches: ";
    cin >> v;

    for (i = 0; i < v; i++)
    {
        visited[i] = 0;
        for (j = 0; j < v; j++)
        {
            a[i][j] = 999;
        }
    }

    cout << "\nEnter the no. of connections: ";
    cin >> e;

    for (i = 0; i < e; i++)
    {
        cout << "Enter the end branches of connections:  " << endl;
        cin >> l >> u;
        cout << "Enter the phone company charges for this connection:  ";
        cin >> w;
        a[l-1][u-1] = a[u-1][l-1] = w;  // Sets the cost for both directions
    }
}
```

- `v` is the number of branches (vertices).
- A loop initializes the `visited[]` array and sets all values in the `a[][]` adjacency matrix to `999`, indicating no connections initially.
- `e` is the number of edges (connections).
- The next loop asks the user to input each connection (from branch `l` to `u`), along with the connection charge `w`, and updates the adjacency matrix for both directions (undirected graph).

`display()` Method

```cpp
void tree::display()
{
    cout << "\nAdjacency matrix:";
    for (i = 0; i < v; i++)
    {
        cout << endl;
        for (j = 0; j < v; j++)
        {
            cout << a[i][j] << "   ";
        }
        cout << endl;
    }
}
```

- Displays the **adjacency matrix** of the graph. It shows how branches are connected and the costs (charges) associated with the connections.

`minimum()` Method (Prim's Algorithm)

```cpp
void tree::minimum()
{
    int p = 0, q = 0, total = 0, min;
    visited[0] = 1;  // Start from the first branch

    for (int count = 0; count < (v - 1); count++)
    {
        min = 999;  // Initialize minimum edge cost as a high value
        for (i = 0; i < v; i++)
        {
            if (visited[i] == 1)
            {
                for (j = 0; j < v; j++)
                {
                    if (visited[j] != 1)  // Find the next unvisited node
                    {
                        if (min > a[i][j])  // Find minimum edge
                        {
                            min = a[i][j];
                            p = i;
                            q = j;
                        }
                    }
                }
            }
        }

        visited[p] = 1;  // Mark the branch as visited
```

```
        visited[q] = 1;  // Mark the branch as visited
        total = total + min;  // Add the minimum cost to the total
        cout << "Minimum cost connection is " << (p + 1) << " -> " << (q + 1) << "  with charge: " <
    }

    cout << "The minimum total cost of connections of all branches is: " << total << endl;
}
```

- **Prim's Algorithm** starts from an arbitrary vertex (here, branch 1).
- The algorithm finds the minimum cost edge that connects a visited branch to an unvisited one.
- It adds the cost of the selected edge to the `total` and marks both the connected branches as visited.
- This is repeated until all vertices are included in the MST (Minimum Spanning Tree).

`main()` Method

```cpp

int main()
{
    int ch;
    tree t;
    do
    {
        cout << "==========PRIM'S ALGORITHM==================" << endl;
        cout << "\n1.INPUT\n \n2.DISPLAY\n \n3.MINIMUM\n" << endl;
        cout << "Enter your choice :" << endl;
        cin >> ch;

        switch (ch)
        {
        case 1:
            cout << "*******INPUT YOUR VALUES*******" << endl;
            t.input();
            break;

        case 2:
            cout << "*******DISPLAY THE CONTENTS********" << endl;
            t.display();
            break;

        case 3:
            cout << "*********MINIMUM************" << endl;
            t.minimum();
            break;
        }

    } while (ch != 4);
    return 0;
}
```

- The `main()` function provides a simple menu to:
  - **Input** the graph (vertices and edges).
  - **Display** the adjacency matrix.
  - **Run Prim's algorithm** to find the MST.

Key Concepts in the Code:

1. **Prim's Algorithm**: This is used for finding the Minimum Spanning Tree (MST) of a graph. It starts with an arbitrary vertex, adding the minimum cost edge that connects a visited vertex to an unvisited one.
2. **Adjacency Matrix**: A 2D matrix that represents a graph. The value at position `a[i][j]` represents the weight of the edge between vertex `i` and vertex `j`.
3. **Visited Array**: This array keeps track of which vertices have been included in the MST.

What This Code Does:

- **Input**: The user enters the number of vertices (branches) and edges (connections), followed by the connections and their associated costs.
- **Display**: It shows the graph in the form of an adjacency matrix.
- **Minimum (Prim's Algorithm)**: It computes and displays the minimum spanning tree using Prim's algorithm, showing each selected edge and the total minimum cost.

Recommended Video:

I suggest checking out this **YouTube video on Prim's Algorithm** for a visual explanation:
Prim's Algorithm | Graph Algorithms | GeeksforGeeks
It walks through the theory and gives step-by-step insights into how the algorithm works.

Let me know if you need further clarification or have any questions!