Copilot

Certainly! Let's explore different ways to represent and implement graphs in C++. Graphs are essential data structures used to model relationships between entities. We'll cover two common representations: **Adjacency Matrix** and **Adjacency List**.

Graph Representation in C++:

1. Adjacency Matrix:

An adjacency matrix is a 2D matrix that represents a graph. It uses boolean values (0s and 1s) to indicate whether an edge exists between two vertices. Let's see how to implement it:

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

const int MAX_VERTICES = 100; // Maximum number of vertices

class Graph {
private:
    int vertices;
    bool adjMatrix[MAX_VERTICES][MAX_VERTICES];

public:
    Graph(int v) : vertices(v) {
        for (int i = 0; i < vertices; ++i) {
            for (int j = 0; j < vertices; ++j) {
                adjMatrix[i][j] = false;
            }
        }
    }

    void addEdge(int src, int dest) {
        adjMatrix[src][dest] = true;
        adjMatrix[dest][src] = true; // For an undirected graph
    }

    void printAdjacencyMatrix() {
        for (int i = 0; i < vertices; ++i) {
            for (int j = 0; j < vertices; ++j) {
                cout << adjMatrix[i][j] << " ";
            }
            cout << endl;
        }
    }
};

int main() {
    int v, e;
    cout << "Enter the number of vertices: ";
    cin >> v;

    Graph G(v);

    cout << "Enter the number of edges: ";
    cin >> e;

    for (int i = 0; i < e; ++i) {
        int src, dest;
        cout << "Enter edge " << i + 1 << " (source and destination): ";
        cin >> src >> dest;
        G.addEdge(src, dest);
    }

    cout << "\nAdjacency Matrix:\n";
    G.printAdjacencyMatrix();
```

```
    return 0;
}
```

2. Adjacency List:

An adjacency list uses an array of lists to store edges between vertices. Each index in the array represents a specific vertex, and the list at that index contains adjacent vertices. Let's implement it:

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

class Graph {
private:
    int vertices;
    list<int>* adjLists;

public:
    Graph(int v) : vertices(v) {
        adjLists = new list<int>[v];
    }

    void addEdge(int src, int dest) {
        adjLists[src].push_back(dest);
        adjLists[dest].push_back(src); // For an undirected graph
    }

    void printAdjacencyList() {
        for (int i = 0; i < vertices; ++i) {
            cout << "V(" << i << ") -> ";
            for (int vertex : adjLists[i]) {
                cout << vertex << " ";
            }
            cout << endl;
        }
    }
};

int main() {
    int v, e;
    cout << "Enter the number of vertices: ";
    cin >> v;

    Graph G(v);

    cout << "Enter the number of edges: ";
    cin >> e;

    for (int i = 0; i < e; ++i) {
        int src, dest;
        cout << "Enter edge " << i + 1 << " (source and destination): ";
        cin >> src >> dest;
        G.addEdge(src, dest);
    }

    cout << "\nAdjacency List:\n";
    G.printAdjacencyList();

    return 0;
}
```

Choose the representation that best suits your use case! 📊🔗