Date: 21/01/2021

Practical 1

2CSDE56 - Graph Theory

Name: Shrey Viradiya

Roll No: 18BCE259

Aim:

Write a C / C++ program to find union, intersection, complement, sum, and difference of two graphs. Use the adjacency list for representing the graph.

Code:

Prac1_AdjList.cpp

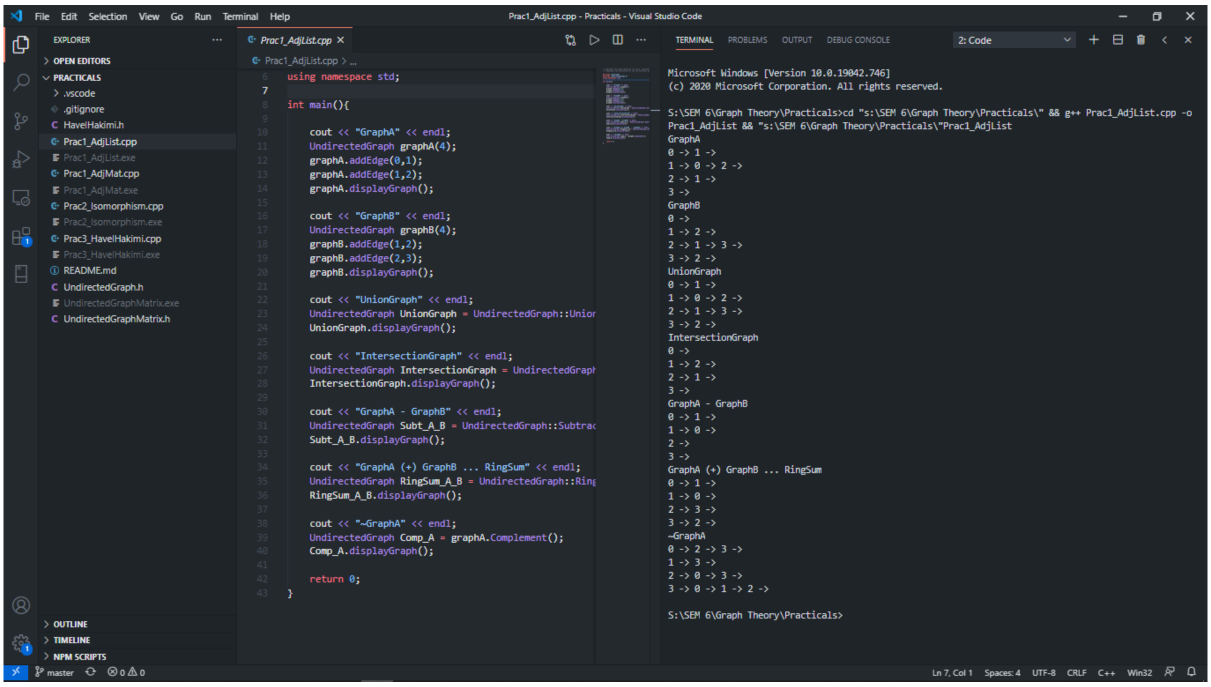
```cpp
// Implement the graph operation for union, intersection,
// compliment and subtraction of the two different graphs

#include <iostream>
#include "UndirectedGraph.h"
using namespace std;

int main(){

    cout << "GraphA" << endl;
    UndirectedGraph graphA(4);
    graphA.addEdge(0,1);
    graphA.addEdge(1,2);
    graphA.displayGraph();

    cout << "GraphB" << endl;
    UndirectedGraph graphB(4);
    graphB.addEdge(1,2);
    graphB.addEdge(2,3);
    graphB.displayGraph();

    cout << "UnionGraph" << endl;
    UndirectedGraph UnionGraph = UndirectedGraph::Union(graphA, graphB);
    UnionGraph.displayGraph();

    cout << "IntersectionGraph" << endl;
    UndirectedGraph IntersectionGraph = UndirectedGraph::Intersection(graphA,
graphB);
    IntersectionGraph.displayGraph();

    cout << "GraphA - GraphB" << endl;
    UndirectedGraph Subt_A_B = UndirectedGraph::Subtraction(graphA, graphB);
    Subt_A_B.displayGraph();

    cout << "GraphA (+) GraphB ... RingSum" << endl;
    UndirectedGraph RingSum_A_B = UndirectedGraph::RingSum(graphA, graphB);
    RingSum_A_B.displayGraph();

    cout << "~GraphA" << endl;
    UndirectedGraph Comp_A = graphA.Complement();
    Comp_A.displayGraph();

    return 0;
}
```

UndirectedGraph.h

```cpp
#pragma once
#include<vector>
#include<set>
#include<iterator>
#include<iostream>
#include<algorithm>
using namespace std;

class UndirectedGraph
{
public:
    int numVertices;
    vector <set <int>> graph;
    UndirectedGraph(int V);
    void addEdge(int src, int dest);
    void displayGraph();
    static UndirectedGraph Union(UndirectedGraph graphA, UndirectedGraph graph
B);
    static UndirectedGraph Intersection(UndirectedGraph graphA, UndirectedGrap
h graphB);
    static UndirectedGraph Subtraction(UndirectedGraph graphA, UndirectedGraph
 graphB);
    static UndirectedGraph RingSum(UndirectedGraph graphA, UndirectedGraph gra
phB);
    UndirectedGraph Complement();
};

UndirectedGraph::UndirectedGraph(int V){
    numVertices = V;
    for (int i = 0; i < numVertices; i++)
    {
        graph.push_back(set<int> {});
    }
}

void UndirectedGraph::displayGraph(){
    int node = 0;
    for (auto i = graph.begin(); i != graph.end(); i++)
    {
        cout << (node++) << " -> ";
        for (auto j = (*i).begin(); j != (*i).end(); j++)
        {
            cout << *j << " -> ";
        }
        cout << endl;
    }
}
```

```cpp
void UndirectedGraph::addEdge(int src, int dest){
    graph[src].insert(dest);
    graph[dest].insert(src);
}

UndirectedGraph UndirectedGraph::Union(UndirectedGraph graphA, UndirectedGraph
 graphB){
    int V = max(graphA.numVertices, graphB.numVertices);
    UndirectedGraph union_graph(V);

    for (int i = 0; i < V; i++)
    {
        set_union(
            graphA.graph[i].begin(),
            graphA.graph[i].end(),
            graphB.graph[i].begin(),
            graphB.graph[i].end(),
            inserter(union_graph.graph[i], union_graph.graph[i].begin())
            );
    }

    return union_graph;
}

UndirectedGraph UndirectedGraph::Intersection(UndirectedGraph graphA, Undirect
edGraph graphB){
    int V = max(graphA.numVertices, graphB.numVertices);
    UndirectedGraph intersection_graph(V);

    for (int i = 0; i < V; i++)
    {
        set_intersection(
            graphA.graph[i].begin(),
            graphA.graph[i].end(),
            graphB.graph[i].begin(),
            graphB.graph[i].end(),
            inserter(intersection_graph.graph[i], intersection_graph.graph[i].
begin())
            );
    }

    return intersection_graph;
}

UndirectedGraph UndirectedGraph::Subtraction(UndirectedGraph graphA, Undirecte
dGraph graphB){
    int V = max(graphA.numVertices, graphB.numVertices);
    UndirectedGraph subtracted_graph(V);

    for (int i = 0; i < V; i++)
    {
        set_difference(
            graphA.graph[i].begin(),
            graphA.graph[i].end(),
            graphB.graph[i].begin(),
            graphB.graph[i].end(),
            inserter(subtracted_graph.graph[i], subtracted_graph.graph[i].begi
n())
            );
    }

    return subtracted_graph;
}

UndirectedGraph UndirectedGraph::RingSum(UndirectedGraph graphA, UndirectedGra
ph graphB){
    return Subtraction(
        Union(graphA, graphB),
        Intersection(graphA, graphB)
        );
}

UndirectedGraph UndirectedGraph::Complement(){
    UndirectedGraph complement_graph(numVertices);

    set <int> allVer;
    for (int i = 0; i < numVertices; i++)
    {
        allVer.insert(i);
    }

    for (int i = 0; i < numVertices; i++)
    {
        allVer.erase(i);
        set_difference(
            allVer.begin(),
            allVer.end(),
            graph[i].begin(),
            graph[i].end(),
            inserter(complement_graph.graph[i], complement_graph.graph[i].begi
n())
            );
        allVer.insert(i);
    }
    return complement_graph;
}
```

Snapshot of the output:



```
using namespace std;

int main(){

    cout << "GraphA" << endl;
    UndirectedGraph graphA(4);
    graphA.addEdge(0,1);
    graphA.addEdge(1,2);
    graphA.displayGraph();

    cout << "GraphB" << endl;
    UndirectedGraph graphB(4);
    graphB.addEdge(1,2);
    graphB.addEdge(2,3);
    graphB.displayGraph();

    cout << "UnionGraph" << endl;
    UndirectedGraph UnionGraph = UndirectedGraph::Unior
    UnionGraph.displayGraph();

    cout << "IntersectionGraph" << endl;
    UndirectedGraph IntersectionGraph = UndirectedGraph
    IntersectionGraph.displayGraph();

    cout << "GraphA - GraphB" << endl;
    UndirectedGraph Subt_A_B = UndirectedGraph::Subtrac
    Subt_A_B.displayGraph();

    cout << "GraphA (+) GraphB ... RingSum" << endl;
    UndirectedGraph RingSum_A_B = UndirectedGraph::Ring
    RingSum_A_B.displayGraph();

    cout << "~GraphA" << endl;
    UndirectedGraph Comp_A = graphA.Complement();
    Comp_A.displayGraph();

    return 0;
}
```

```
Microsoft Windows [Version 10.0.19042.746]
(c) 2020 Microsoft Corporation. All rights reserved.

S:\SEM 6\Graph Theory\Practicals>cd "s:\SEM 6\Graph Theory\Practicals\" && g++ Prac1_AdjList.cpp -o
Prac1_AdjList && "s:\SEM 6\Graph Theory\Practicals\"Prac1_AdjList
GraphA
0 -> 1 ->
1 -> 0 -> 2 ->
2 -> 1 ->
3 ->
GraphB
0 ->
1 -> 2 ->
2 -> 1 -> 3 ->
3 -> 2 ->
UnionGraph
0 -> 1 ->
1 -> 0 -> 2 ->
2 -> 1 -> 3 ->
3 -> 2 ->
IntersectionGraph
0 ->
1 -> 2 ->
2 -> 1 ->
3 ->
GraphA - GraphB
0 -> 1 ->
1 -> 0 ->
2 ->
3 ->
GraphA (+) GraphB ... RingSum
0 -> 1 ->
1 -> 0 ->
2 -> 3 ->
3 -> 2 ->
~GraphA
0 -> 2 -> 3 ->
1 -> 3 ->
2 -> 0 -> 3 ->
3 -> 0 -> 1 -> 2 ->

S:\SEM 6\Graph Theory\Practicals>
```