OOAIA Lab-3: Graph Operations with Operator Overloading

Challenge Link: link

Problem Statement

Implement a `Graph` class that represents an undirected graph and supports various operations using operator overloading.

Methods to be Implemented

1. **`operator+`**: Union of two graphs
2. **`operator-`**: Intersection of two graphs
3. **`operator!`**: Complement of a graph
4. **`operator>>`**: Input a graph
5. **`operator<<`**: Output a graph
6. **`isReachable`**: Check if there's a path between two vertices
7. **`addEdge`**: Add an edge between two vertices
8. **`removeEdge`**: Remove an edge between two vertices

Formal Definitions

Union of Graphs (G1 + G2)

Let G1(V1, E1) and G2(V2, E2) be two graphs. The union of G1 and G2 is a graph G = G1 ∪ G2, where:

- Vertex set V = V1 ∪ V2
- Edge set E = E1 ∪ E2

Intersection of Graphs (G1 - G2)

Let G1(V1, E1) and G2(V2, E2) be two graphs. The intersection of G1 and G2 is a graph G = G1 ∩ G2, where:

- Vertex set V = V1 ∪ V2

- Edge set E = E1 ∩ E2

Complement of a Graph (!G)

Let G(V, E) be a simple graph and K be the set of all 2-element subsets of V. The complement of G, denoted as G', is defined as: G' = (V, K \ E) Where K \ E is the relative complement of E in K. In other words, the complement graph has:

- The same vertex set V as the original graph
- An edge between two vertices if and only if there is no edge between them in the original graph

Important Notes

1. It is **compulsory** to use operator overloading for implementing union (+), intersection (-), complement (!), input (<<) and output (>>).
2. The graph uses **0-based indexing** for vertices.
3. The graph is undirected, meaning an edge (u, v) is the same as (v, u).

Input Format

The input consists of multiple operations:

1. First line: `Graph`
2. Second line: `N M` (N = number of vertices, M = number of edges)
   - 1 ≤ N ≤ 10^3
   - 0 ≤ M ≤ min(N * (N-1) / 2, 10^5)
3. Next M lines: `u v` (representing an edge between vertices u and v)
4. Subsequent lines: Various operations as described below

Operations

- `union`: Followed by another graph definition (using the overloaded >> operator)
- `intersection`: Followed by another graph definition (using the overloaded >> operator)
- `complement`
- `isReachable u v`: Check if vertex v is reachable from vertex u
- `add_edge u v`: Add an edge between vertices u and v. If the edge already exists, do nothing.
- `remove_edge u v`: Remove the edge between vertices u and v. If the edge doesn’t exist do nothing.

- **`printGraph`**: Display the current state of the graph (using the overloaded << operator)
- **`end`**: Terminate the program

Output Format

- For **`isReachable`**: Print "Yes" if reachable, "No" otherwise
- For **`printGraph`**: Use the overloaded << operator to display each vertex and its adjacent vertices
- For other operations: No output unless specified

Sample Input

```
Graph
4 3
0 1
1 2
2 3
printGraph
isReachable 0 3
complement
printGraph
end
```

Sample Output

```
Vertex 0: 1
Vertex 1: 0 2
Vertex 2: 1 3
Vertex 3: 2
Yes
Vertex 0: 2 3
Vertex 1: 3
Vertex 2: 0
Vertex 3: 0 1
```

Implement the `Graph` class and the `main()` function to handle these operations efficiently.