Implement Graph Data Structure

Lab-3: Graph Operations with Operator Overloading

Problem Statement

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

For better readability: link

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, where V is the set of vertices and E is the set of edges.

The complement of G, denoted as G' = (V, E'), is defined as follows:

1. G' has the same set of vertices V as G.
2. For any two distinct vertices u and v in V:
   - (u, v) is an edge in E' if and only if (u, v) is not an edge in E.

In other words: E' = {(u, v) | u, v ∈ V, u ≠ v, and (u, v) ∉ E}

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)
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
- `remove_edge u v`: Remove the edge between vertices u and v
- `printGraph`: Display the current state of the graph (using the overloaded << operator)
- `end`: Terminate the program

Constraints

```
- 1 ≤ N ≤ 10^3 for general operations
- 0 ≤ M ≤ min(N * (N-1) / 2, 10^5)
```

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 0

```
Graph 3 2
0 1
1 2
union
Graph 3 3
0 1
2 0
1 2
printGraph
isReachable 0 2
remove_edge 2 0
remove_edge 2 0
remove_edge 2 1
remove_edge 0 2
isReachable 0 2
printGraph
complement
printGraph
end
```

Sample Output 0

```
Vertex 0: 1 2
Vertex 1: 0 2
Vertex 2: 0 1
Yes
No
Vertex 0: 1
Vertex 1: 0
Vertex 2:
Vertex 0: 2
Vertex 1: 2
Vertex 2: 0 1
```

Sample Input 1

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

Sample Output 1

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