EC327 - Spring 2025 - Lab 5

Background

This lab assignment will provide hands-on experience with fundamental object-oriented concepts, focusing on overloading operators and standard libraries.

Important

Please make sure your code compiles and runs as intended on the engineering grid. Code that does not compile will NOT be graded and will receive a 0.

Submission Instructions

You will submit this assignment as a single .zip file with the following name:

<first-name>_<last-name>-lab5.zip

So for example:

ed_solovey-lab5.zip.

Note on Collaboration

You are welcome to talk to your classmates about the assignment and discuss high level ideas. However, all code that you submit must be your own. We will run code similarity tools against your submissions and will reach out with questions if anything is flagged as suspicious.

The closed-book exams for this class will mostly cover material very similar to what you do on the homework assignments. The best way to prepare for the exams is to do the assignments on your own and internalize the learnings from that process. Cheating on the assignments will very likely result in you not doing well on the exams.

Helpful Information

In this lab, we hope you can start to become familiar with how to use `std::logic_error`.

In C++, a `std::logic_error` is one of the standard exception types provided by the language that signals an error in the *program's logic*—such as invalid assumptions, incorrect use of function parameters, or other mistakes that do not depend on external factors (like user input or files).

A minimal structure for exception handling in C++ looks like this:

```
try {
   // Code that might throw an exception
   if (someErrorCondition) {
       throw std::logic_error("Something went wrong!");
   }
}
catch (const exception& e) {
   // Handle the exception (e.g., log it, display a message, recover)
   std::cerr << "Error: " << e.what() << std::endl;
}
```

Below is an example demonstrating how to throw a `std::logic_error` in a Fibonacci function when the input is negative.

```
#include <stdexcept>
#include <iostream>

using namespace std;

int fibonacci(int n) {
   if (n < 0) {
       throw logic_error("Fibonacci is not defined for negative integers.");
   }
   if (n == 0) {
       return 0;
   }
   if (n == 1) {
       return 1;
   }
   return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```cpp
int main() {
    try {
        // Code that might throw an exception
        cout << "fib(5) = " << fibonacci(5) << endl;
        cout << "fib(-2) = " << fibonacci(-2) << endl; // Will throw an error
    } catch (const logic_error& e) {
        // Handle the exception (e.g., log it, display a message, recover)
        cerr << "Logic error encountered: " << e.what() << endl;
    }
    return 0;
}
```

If you want to learn more about exception in C++ please visit this link here or here.

Problem 1 - Graph Class(60 points)

Submission Instructions

Your solution to this problem should contribute a **cpp** file and a **header** file to your overall **lab5** zip. The **cpp** and **header** file should follow the following format:

<first-name>_<last-name>-lab5-1.cpp
lab5_problem1.h

So for example, my file would be:

ed_solovey-lab5-1.cpp
lab5_problem1.h

Actual Problem

Create a class called with the following contents:

```
#ifndef LAB5_PROBLEM1_H
#define LAB5_PROBLEM1_H

#include <map>
#include <set>
#include <stdexcept>
#include <iostream>
using namespace std;

/**
* A graph is a collection of vertices, together with edges connecting
* some of the vertices.
*
* Implement an undirected Graph class with the following properties:
* -  There is only one public default constructor that produces an
*    empty graph with no vertices (or edges).
* -  operator+= is used to add a vertex with a given ID to the graph.
*       if a vertex already has the given ID, a logic_error exception
*       should be thrown.
* -  hasEdge(int ID1, int ID2) returns true iff an edge between vertices
*       with ids ID1 and ID2 exists.
```

```
* -  addEdge(int ID1, int ID2) adds an edge between vertices with
*       ids ID1 and ID2, respectively.
*       if one of the IDs does not exist, or the edge already exists,
*       an invalid_argument exception should be thrown.
* -  numVertices() returns the number of vertices in the graph
* -  numEdges() returns the number of edges in the graph
*
* @example
* Graph G;
* G+=3;
* G+=5;
* G+=3;            // throws logic_error
* G.hasEdge(3,5);  // returns false
* G.addEdge(5,3);  // adds an edge connecting vertex 5 and 3.
* G.hasEdge(3,5);  // returns true (edge between 5 to 3 == between 3 and 5).
* G.numVertices(); // returns 2 - vertex ID 3 and ID 5
* G.numEdges();    // returns 1 - between vertex ID 3 and ID 5
*/
class Graph {
   private:
       // create your own private variables and functions here if needed

   public:
       Graph& operator+=(int ID);
       bool hasEdge(int ID1, int ID2);
       void addEdge(int ID1, int ID2);
       int numVertices();
       int numEdges();
       /**
       * Create your public variables and functions that you think will be
       * useful(shouldn't need any public functions, all helper functions and
       * variables should be private)
       */
};
#endif //LAB5_PROBLEM1_H
```

Hint: Thinking about how to use std::map<int, std::set<int>>.

You can test your implementation from a **main** function or from tests you set up otherwise. Your submission will be tested against multiple inputs, and you should convince yourself that your

solution works for the general case. This goes for all the problems. When submitting your code please do not include any main functions.

Problem 2 - Two Hop Neighbors (40 points)

Submission Instructions

Your solution to this problem should contribute a **cpp** file and a **header** file to your overall **lab5** zip. The **cpp** and **header** file should follow the following format:

<first-name>_<last-name>-lab5-2.cpp
lab5_problem2.h

So for example, my file would be:

ed_solovey-lab5-2.cpp
lab5_problem2.h

Actual Problem

For this question, you need to extend your **Graph** class defined in question 1. Copy your working problem1 implementation into your **.cpp** file. Create a header file called **lab5_problem2.h** with the following contents:

```
#ifndef LAB5_PROBLEM2_H
#define LAB5_PROBLEM2_H

#include <set>
using namespace std;

/**
* Copy all your code from lab5_problem1.h to here.
*
* Implement a twoHopNeighbors(int ID) member function in your existing
* Graph class that returns a set of all vertices that can be
* reached by exactly two hops from the given nodevID. Specifically, you
* will copy all of your code from lab5_problem1.h into lab5_problem2.h
* and add this new function. Two-hop neighbors are defined as the
* neighbors of the neighbors of ID, *excluding ID itself*. and they
* must be stored in an std::set (ensuring the final output is sorted).
* If ID does not exist or has no two-hop neighbors, return an empty set.
*
```

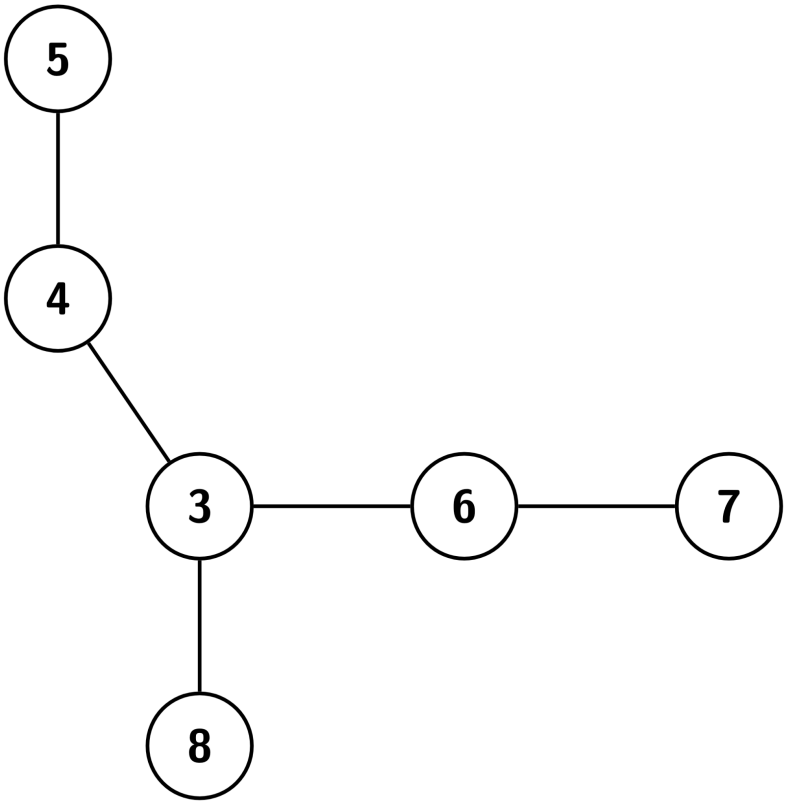
```
* @example
* Graph G;
* G+=3;
* G+=4;
* G+=5;
* G+=6;
* G+=7;
* G+=8;
* G.addEdge(3,4);
* G.addEdge(4,5);
* G.addEdge(3,6);
* G.addEdge(6,7);
* G.addEdge(3,8);
*
* G.twoHopNeighbors(3); // returns {5, 7}
* G.twoHopNeighbors(4); // returns {6, 8}
* G.twoHopNeighbors(5); // returns {3}
* G.twoHopNeighbors(6); // returns {4, 8}
* G.twoHopNeighbors(7); // returns {3}
*
* @example
* Graph G;
* G+=3;
* G+=4;
* G+=9;
* G+=6;
* G.addEdge(3,4);
* G.addEdge(4,9);
* G.addEdge(3,9);
* G.addEdge(9,6);
* G.addEdge(3,6);
*
* G.twoHopNeighbors(3); // returns {4, 6, 9}
* G.twoHopNeighbors(4); // returns {3, 6, 9}
* G.twoHopNeighbors(9); // returns {3, 4, 6}
* G.twoHopNeighbors(6); // returns {3, 4, 9}
*/
class Graph {
   private:
       // all your variables from question 1 and any helper functions you need

   public:
```

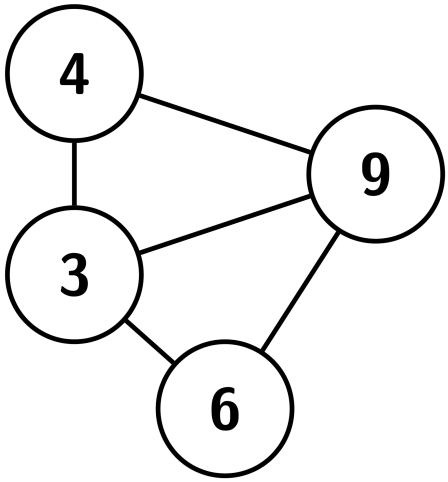
```
        //Problem 1
        Graph& operator+=(int ID);
        bool hasEdge(int ID1, int ID2);
        void addEdge(int ID1, int ID2);
        int numVertices();
        int numEdges();
        //Problem 2
        set<int> twoHopNeighbors(int ID);
        /**
        * Create your public variables and functions that you think will be
        * useful(shouldn't need any public functions, all helper functions and
        * variables should be private)
        */
};

#endif //LAB5_PROBLEM2_H
```

The example graph of the given example 1 can be found here:



The example graph of the given example 2 can be found here:



Your **<first-name>_<last-name>-lab5-2.cpp** should implement the above **Graph::neighborNodes** function.