CSCI 2270: Data Structures - **Final Exam**
Part II (Coding Problems)

Summer 2021: Friday, July 23

Instructions

1. This portion of the exam has two coding questions. Each question is worth 30 points.
2. You are provided with starter code for both questions. For both questions, prototypes are provided.
3. Make sure to use the C++11 standard for compilation (`g++ -std=c++11`). If you have doubts or issues with your local system, compile in `coding.csel.io`.
4. You need to write code for the function asked in the question. You can write helper functions if desired. You do not have to complete the entire class implementation. For example, if the question asks to sum all the nodes in a Linked list, you should only write the code for that function. You do not have to be concerned about other class member functions, such as insertion or list creation.
5. Read the specification given in the questions. Your code should be well-written in terms of commenting and indentation. Good coding includes (but not limited to):
   - meaningful names of the functions and variables,
   - proper indentations, and
   - legible comments.
6. Describe your logic clearly in coding comments. In case your implementation is not fully correct, your comments may be helpful to get you some partial credit.
7. You are required to upload a single zip file with the two sub-directories (folders): `Q1` and `Q2`. Include all of the relevant code (including any starter code) for question 1 and question 2 to the folders `Q1` and `Q2`, respectively. When finished, submit the **combined** zip file (that has both aforementioned folders) to the specified Canvas submission link.
8. In addition to being tested on several test-cases, the grading staff will also visually inspect your code for correctness, partial correctness, and memory leaks.

1. $\boxed{\text{30 points}}$ **Who's the boss?** You are given an implementation of a directed and unweighted graph class (`Graph.cpp, Graph.hpp`) and a driver file (`Driver.cpp`). Please familiarize yourself with the code.

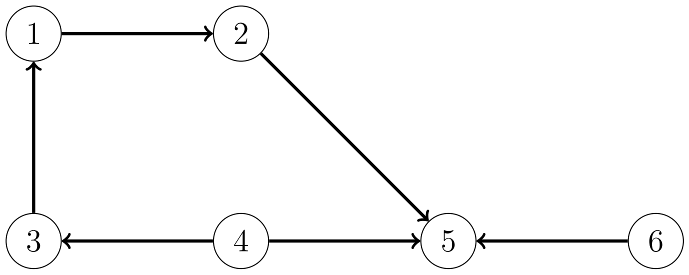
   We say that a given vertex of a Graph is a *boss vertex*, if there exists a path from that vertex to every other vertex in the graph. Similarly, we say that a graph is a boss graph if it contains at least one boss vertex.

   You are required to implement two member functions `isVertexABoss` and `isGraphABoss`. The first function, `isVertexABoss`, returns `true` if the given input vertex is a *boss* vertex, and `false` otherwise. Similarly, the function `isGraphABoss` should return `true` if the graph is a boss graph, and `false`, otherwise.

   The prototypes of your functions must be:

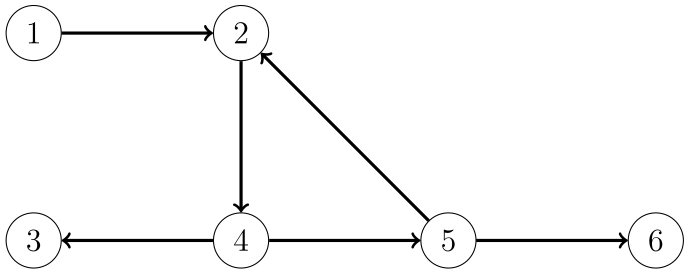
```
    bool isVertexABoss(vertex *v);
    bool isGraphABoss();
```

Example A. Consider the following graph:



Since there is no vertex that can reach all other vertices, the function `isGraphABoss()` must return `false`. Moreover, `isVertexABoss()` should return `false` when called on any vertex of this graph.

Example B. Now consider the following graph:



Notice that from the vertex `"1"` there is a path to every other vertex in the graph. Hence, `isVertexABoss()` should return `true` when called on the vertex `"1"`. Also, for the same reason the function `isGraphABoss()` must return `true`.

Notes. You must implement your code within the given C++ Graph class implementation. Your code must compile with the following command in a standard Linux or Mac terminal.

```
g++ -std=c++11 Graph.cpp Driver.cpp
```

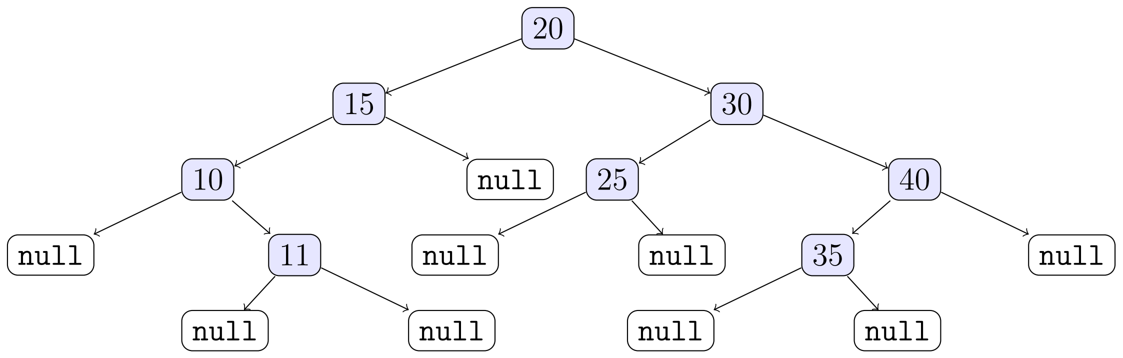
2. 30 points **Look Whooo's counting?**

For this question, you are given an implementation of a binary search tree (BST) class (`bst.cpp`, `bst.hpp`) and a drive file (`main.cpp`). Please familiarize yourself with the code.

You are required to implement a search function `searchCounter` that returns the total number of comparisons to search if a node containing the target key is in the BST. Your function takes the search key value (of type `int`) and returns the total number of comparisons. The function prototype provided in the starter code is:

```
int searchCounter(int target);
```

Examples. Consider the following BST:



A call to `searchCounter(11)` should return `4`, while a call to `searchCounter(20)` should return `1`. Note that if the target value is not found in the BST, the `searchCounter()` function still returns the number of comparisons. For instance, `searchCounter(38)` should return 4, while `searchCounter(16)` should return 2.

Notes. You must implement your code within the given C++ BST class implementation. Your code must compile with the following command in a standard Linux or Mac terminal.

```
g++ -std=c++11 bst.cpp main.cpp
```