CSCI 2270 - Data Structures Name_____

Instructor: Hoenigman/Zagrodzki/Zietz Recitation time_____

Recitation TA_____

Midterm 2

April 19, 2018

You are allowed to use any materials on this exam that you have stored on your own computer or USB drive. Acceptable materials include notes or assignment solutions downloaded from Moodle prior to the exam, notes from lecture or any other notes you have written or typed, and your own solutions to assignments and recitations.

Anything other than permitted materials, such as texts from friends, Internet searches, email, etc is not permitted and is considered cheating. Sharing information about the exam with other people in the class before everyone has had a chance to take the exam is also considered cheating.

If you are caught cheating, it guarantees you an F in the class, a date with the Honor Code council, and my animosity for all time.

So, please sign: "On my honor as a University of Colorado student, I promise not to cheat on this test by giving or receiving help from my peers, or engaging in any other activities that are not allowed under the rules of the exam."

_____

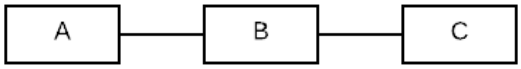
Now, on to the exam…

Part 1 - Programming

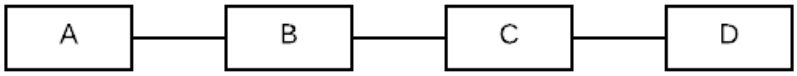
For Part 1 of this exam, you need to use the code provided on Moodle and complete the following two questions. If you haven't already done so, download the Moodle code, called Midterm 2 makeup starter code. There are three files in the .zip file: Midterm2Student.cpp, BST.cpp, and Graph.cpp. To build the code in Sublime, click Build from the Midterm2Student.cpp file and it should build and run the code.

1. Write a method in the Graph class to determine if two vertices share an adjacent vertex, but are not connected to each other.

For example, in this small graph, A and C share an adjacent vertex B.



However, in this graph, A and D do not share an adjacent vertex because A is only adjacent to B and D is only adjacent to C. There would need to be an edge added between A-C, or B-D for A and D to share an adjacent vertex.



Your method should be called *studentFunction*, and take two string arguments, and return a bool:

```
bool Graph::studentFunction(string v1, string v2)
```

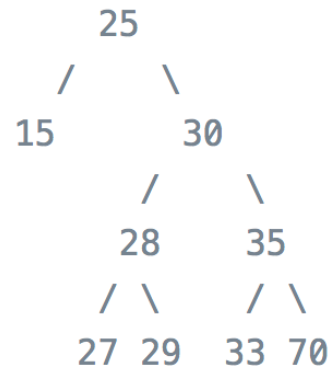
The input arguments are the key values for the vertices, and then function returns true if the vertices share an adjacent vertex and false otherwise. In the code provided, there is a graph already built. Use that graph to verify that your code passes the following test cases.

```
//returns false since a adjacent to b
cout << g.studentFunction("a", "b") << endl;

//returns true since a-d-c in graph, a and c both adjacent to d
cout << g.studentFunction("a", "c") << endl;

//returns false since a-d-e-g in graph is path a - g
cout << g.studentFunction("a", "g") << endl;

//returns false since start and end vertex are the same
cout<<g.studentFunction(“a”,”a”)<<endl;

//returns false because a adjacent to d, even though a-e-d is
//also a path where they share an adjacent vertex
cout<<g.studentFunction(“a”,”d”)<<endl;
```

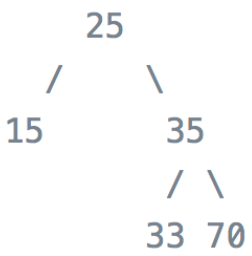
2. Add a method to the BST class that deletes a given value from the BST. The method takes one argument, the value of the node to be deleted. Replace the node to be deleted with its right child. Move the left subtree of the deleted node to the appropriate position in the replacement node's right subtree, only if the replacement has a right subtree, to maintain the BST properties.

   Use the following function header:
   ```
   void BST::studentFunction(int value)
   ```

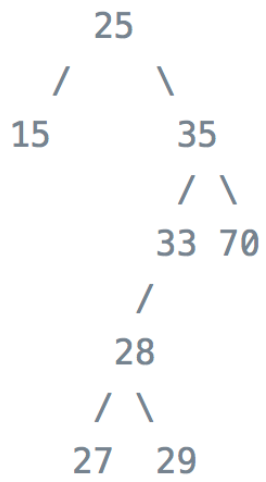
   For example, if you are given the following tree:



   and the value 30, you first need to delete the 30 and replace it with the subtree rooted at 35



   Then, re-insert the subtree for the 30's left child as the left child of the 33, the only place it could go to be a valid BST. If the 35 didn't have a left child, the 28 would have been its new left child.



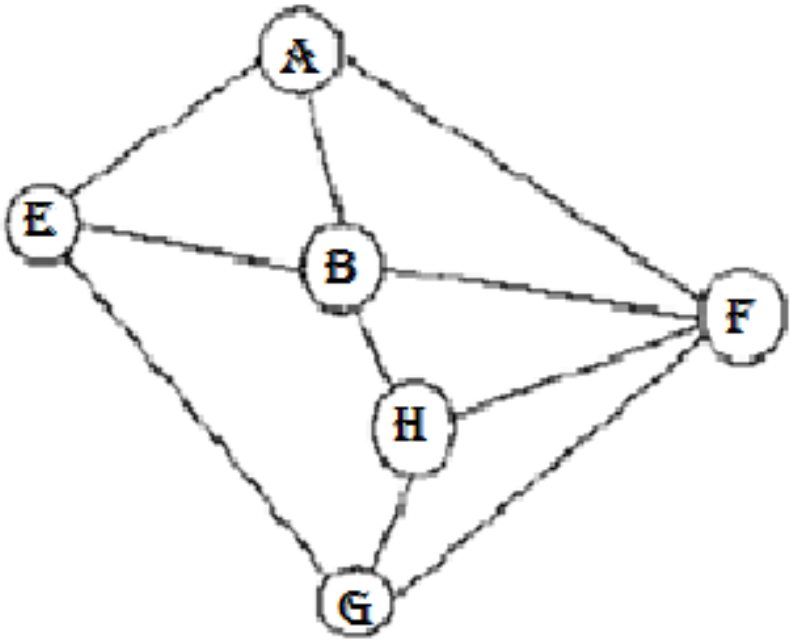
Test cases

Test your code using the following cases

```
bst.studentFunction(78);
inOrderTraversal();
should print 11 12 16 20 21 22 30 34 35 40 45 60 65 70 73 80 90 100 120

bst.studentFunction(30);
inOrderTraversal();
should print 11 12 16 20 21 22 34 35 40 45 60 65 70 73 80 90 100 120

bst.studentFunction(60);
inOrderTraversal();
should print 11 12 16 20 21 22 34 35 40 45 65 70 73 80 90 100 120
```

Part II – Conceptual Questions

1. Consider the graph below:



Which of the following statements is true?

Select one:

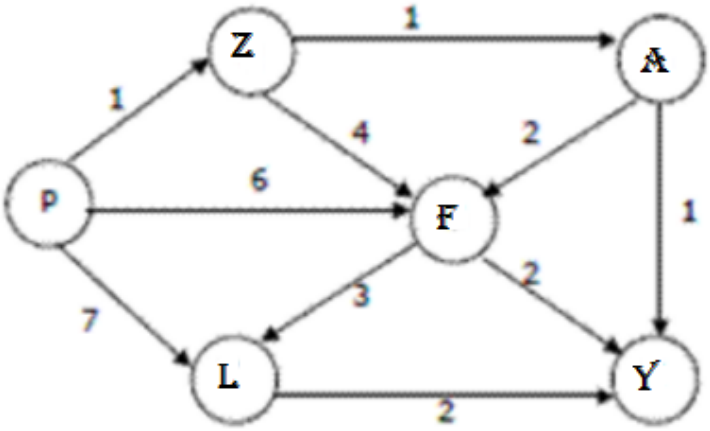
a. Nodes could be visited in a breadth-first search in the order: A B E G H F

b. Nodes could be visited in a breadth-first search in the order: A B H G F E

c. Nodes could be visited in a depth-first search in the order: G E F H B A

d. Nodes could be visited in a depth-first search in the order: G H B A F E

2. Suppose we run Dijkstra's shortest-path algorithm on the following weighted, **directed** graph with vertex **P** as the start vertex and **L** as the end vertex. In what order do the vertices **get solved**? Start your answer with P and list the vertices in the order they are solved. Include the starting vertex and break ties alphabetically. Your answer should be in all capital letters.



3. What is the complexity of search operation in a balanced binary search tree ?

   a. O(log n)

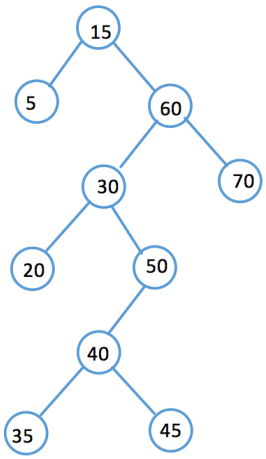
   b. O(n)

   c. O(2^n)

   d. None of these

   e. O(n^2)

4. Given the binary search tree below, show the order that nodes are evaluated in searching for a value of **45** when starting from the root of the tree. You do not need to include any node that is NULL. Write your answer in the order that the nodes are evaluated, separating each node value with a space, e.g., 14 12 11, et cetera.



Order of nodes visited:

Hand this paper back to your exam proctor before you leave the room. We will only grade your exam if we have this paper.

Good luck!