CSCI 2270: Data Structures - Midterm II
Part II (Coding Problems)

Spring 2020: Friday April 10, 5 PM

Instructions (Read it thoroughly)

1. This portion of the exam has three coding questions, two of which need to be completed. Each question is worth 35 points. Question 1 is mandatory. You can choose between question 2 and 3. There is no extra credit in solving all three.

2. For the coding questions, 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 documentation.

3. 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 insertion and list creation.

4. Describe your logic clearly either in coding comments. In case your implementation is not fully correct, your comments may be helpful to get you partial credit.

1. [35 points] Recall the Binary Search Tree (BST) property: If x and y are nodes, and

   1. y is in the left sub-tree of x, then

   $$y.key < x.key$$

   2. y is in the right sub-tree of x, then

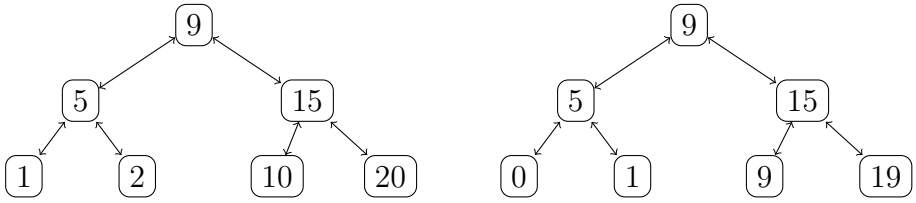
   $$y.key \geq x.key$$

   You are required to write a C++ function `decrementLeaves` that traverses the BST and decrements the value of each leaf node by 1. Moreover, if such a decrement operation causes violation of the BST property, then you need to delete the violating node.

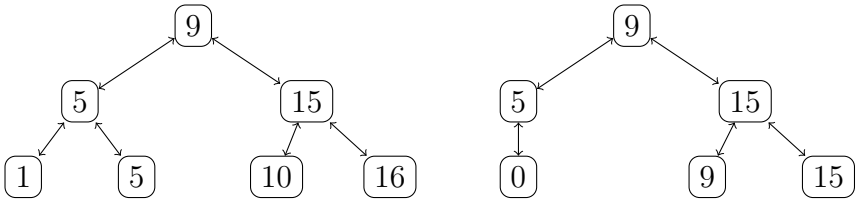
   Implementation strategy tips.

   - You are given the `addNode` function and a sample array in main so you can build and test your tree.
   - Code in stages. First, get the key value decrementing working (not a bad idea to save a backup version of your code at this point.) Then proceed to detecting and deleting an offending node.

   Examples.
   A. For instance, given the BST on the left side, `decrementLeaves` should result in the BST on the right.



   B. As an example of the case that requires node deletion, consider the BST on the left side. The `decrementLeaves` should result in the BST on the right.



2. 35 points A directed graph is said to be *strongly connected* if every vertex is reachable from every other vertex, that is, from any given vertex a path exists to every other vertex in the graph.

   You are required to implement a C++ function to check whether a directed graph is strongly connected. Assume the following structure definitions:

```
struct vertex;

struct adjVertex{
    vertex *v;
};

struct vertex{
    string key;
    bool visited = false;
    int distance = 0;
    vertex *pred = NULL; // predecessor
    std::vector<adjVertex> adj;
};

class Graph{
    private:
        std::vector<vertex*> vertices;

    public:
      bool isStronglyConnected ();
    ...
      // assume all other methods are also implemented

};
```

   You need to implement the following function:

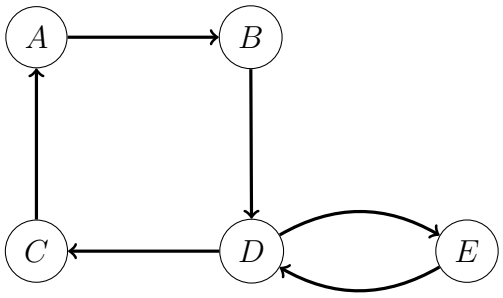
```
bool Graph::isStronglyConnected(){

...
}
```

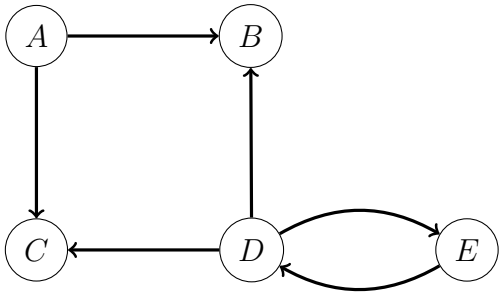
   Note that:

   - `Vertices` is a private member of the class
   - `isStronglyConnected()` is a member function of the Graph class.
   - The graph is directed.

Examples.
A. Graph depicted below is strongly connected graph as for any pair of vertices there is a path.



B. However, the graph shown below is not strongly connected. For example, there is no path from vertex C to any other vertex.



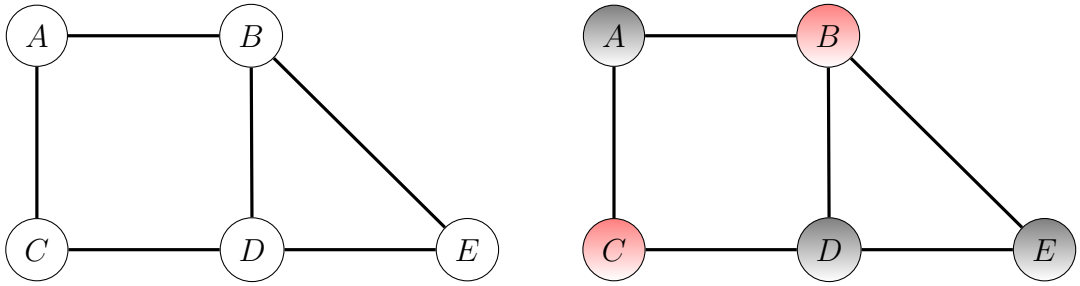
You can use a helper function(s) if you desire.

3. [35 points] Consider an undirected graph with an updated vertex struct definition that includes a member called "color" of type string. Write a function `colorGraph()` that, given a starting vertex, traverses the graph and updates the colors of all the vertices in the following manner:

   1. the starting vertex is "black",
   2. vertices adjacent to the starting vertex are "red",
   3. vertices, with minimum distance 2 from the starting vertex, are "black",
   4. vertices, with minimum distance 3 from the starting vertex, are "red", and so on.
   5. In particular, all of the vertices whose minimum distance from the starting vertex is even are "black", and if it is odd then they are colored "red".
   6. Moreover, if a vertex is not reachable, then its color should be "white".

   Note: you can add a member to the vertex struct definition given in the starter code.

Example A. `colorGraph`(A) on the undirected graph on the left, colors the graph is such a way to produce the graph on the right.



Example B. `colorGraph`(A) on the undirected graph on the left, colors the graph is such a way to produce the graph on the right.