CSCI 104 – Homework 7

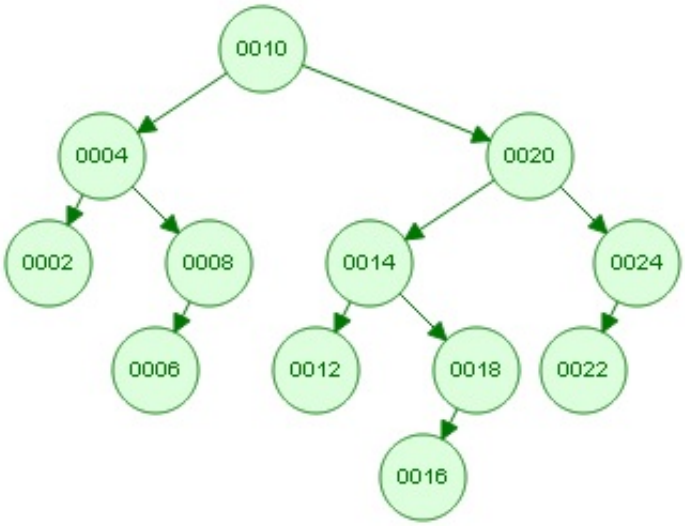
Data Structures and Object Oriented Design

Homework 7

- Directory name for this homework (case sensitive): `hw7`
  - This directory should be in your `hw_username` repository
  - This directory needs its own `README.md` file
  - You should provide a `Makefile` to compile your code for each problem.

Problem 1 (AVL Trees, 10%)

Consider the following initial configuration of an AVL Tree:



Draw the tree representation of the AVL tree after each of the following operations, using the method presented in class (when deleting, always promote a value from the left subtree). Your operations are done in **sequence**, so your tree should have 12 values in it when you're done. Make sure to clearly indicate each of your final answers.

- Insert 15
- Insert 17
- Insert 11
- Remove 20
- Remove 16
- Remove 4

We highly recommend you try to solve these by hand before using any tools to verify your answers.

Problem 2 (Binary Search Tree Iterators, 20%)

We are providing for you a file `bst.h` (in the homework-resources repository) which implements a simple binary search tree. You will need to implement the iterator, so that it traverses the tree using an in-order traversal.

You may add any (private) helper functions you like. A successor helper function (which returns the next largest value after the current node) may be particularly useful.

Problem 3 (AVL Trees, 35%)

We are providing you a half-finished file `avlbst.h` (in the homework-resources repository) for implementing an AVL Tree. It builds on the file you completed for the previous question.

Complete this file by implementing the `insert()` and `remove()` functions for AVL Trees. You are strongly encouraged to use private/protected helper functions.

Note 1: When writing a class (AVLTree) that is derived from a templated class (BinarySearchTree), accessing the inherited members must be done by scoping or preceding with the keyword `this->`.

Note 2 (that may not make sense until you have started coding): Your AVLTree will inherit from your BST. This means the root member of BST is a Node type pointer that points to an AVLNode (since you will need to create AVLNodes with `height` values for your AVLBST), which is fine because a base Node pointer can point at derived AVLNodes. If you need to call AVLNode-specific functions (i.e. members that are part of AVLNode but not Node) then one simple way is to downcast your Node pointer to an AVLNode pointer, as in `static_cast<AVLNode<K,V>*>(root)` to temporarily access the AVLNode-specific behavior/data.

Problem 4 (Exam Scheduling, 35%)

Write a program to schedule exam time slots so that no student has two exams at the same time, using recursion and backtracking. The input file (whose name will be passed in at the command line) will have three parameters on the first line: how many exams there are, how many students there are, and how many timeslots there are, separated by an arbitrary amount of whitespace. Each successive line will have the name of a student (a string of lowercase letters), followed by the name of the classes that student is in (each one being an integer greater than 0). There will be an arbitrary amount of whitespace between the student name and each class. You may assume you always receive a correctly formatted input file.

```
5 4 3
aaron    104 170
michael 104   170 350
```

```
jarjar  101
finn  270  350
```

The output should be an assignment of classes to the integers between `1` and the number of timeslots, one class per line:

```
101 1
104 1
170 2
270 1
350 3
```

There are of course other solutions, you only need to find one that works. If there is no solution for the requested number of timeslots, you should output `No valid solution.`

You must maintain a map of each class to its timeslot, using your AVL implementation. As there is no method to update a value in your AVL Tree, you will need to delete the old value and insert the new value. You may use the STL map instead of your AVL Tree at cost of a 10 point deduction, in case you cannot finish your AVL Tree.

Since some of the function calls throw exceptions, make sure to use `try` and `catch` appropriately, even if you do not expect the catch blocks to be used.

Commit then Re-clone your Repository

Be sure to add, commit, and push your code in your `hw7` directory to your `hw_usc-username` repository. Now double-check what you've committed, by following the directions below (failure to do so may result in point deductions):

1. Go to your home directory: `$ cd ~`
2. Create a `verify` directory: `$ mkdir verify`
3. Go into that directory: `$ cd verify`
4. Clone your hw_username repo: `$ git clone git@github.com:usc-csci104-spring2016/hw_usc-username.git`
5. Go into your hw7 folder `$ cd hw_username/hw7`
6. Recompile and rerun your programs and tests to ensure that what you submitted works.

•