24-783 Problem Set 5

The following instructions assumes that your directory structure is:

(User Directory)
    24783
        src
            CMakeLists.txt (Top-level)
            public
            (Your Andrew ID)

Also in the instruction, a forward slash (/) is used as the directory separator. In Windows command prompt, you need to use a back slash (\) instead.

Two-week assignment. See Canvas for deadline.

Preparation: Set up CMake projects for bitmap and utility libraries and ps5 executable

You first create projects for the two problem sets

In the command line window change directory to:

~/24783/src/*yourAndrewId*

1. Update the course_files repository by typing:
   svn update ~/24783/src/course_files
2. Create a sub-directory called:
   ps5
   and then, inside ps5 create sub-directories:
   ps5_1
   ps5_2
   File/Directory names are case sensitive. Use underscore. NOT hypen.
3. Use "svn copy" to copy the binary-tree class source files explained in class to your ps5 directory. The files are in course_files. The command you type is (in ps5 directory):
   `svn copy ~/24783/src/course_files/bintreeutil bintreeutil`

4. The directory structure under your SVN directory is important. The grading script expects that the directory structure under your SVN directory is:

    ps5

        ps5_1

        ps5_2

        bintreeutil

    (By the way, don't delete ps3, ps4, and simplebitmap directories. Those haven't been graded yet.)
5. Copy main.cpp of the binary-tree visualizer:

    ~/24783/src/course_files/bintreevis/main.cpp

    to ps5_1 and ps5_2 directories.
6. Write CMakeLists.txt for bintreeutil subdirectory. The library name must be "bintreeutil". The library uses bintree.h and bintree.cpp. (The cpp file is empty, but CMake requires at least one .C/.CPP file for a library.)
7. Write CMakeLists.txt for ps5_1, and ps5_2 sub-directories. The project is a graphical application. Therefore use MACOSX_BUNDLE keyword. The project names must be "ps5_1" and "ps5_2". Case sensitive and use underscore. Do not use hyphen. It must link "fslazywindow", "ysbitmapfont", and "bintreeutil" libraries.
8. Modify top-level CMakeLists.txt so that your build tree includes bintreeutil, and ps5/ps5_1 and ps5/ps5_2 sub-directories.
9. Run CMake, compile, and run ps5_1 and ps5_2 executables.
10. Add all the files you created to the control of svn. Svn-copied files are already under SVN's control, and you don't have to add them.
11. Commit to the SVN server.
12. Check out your directory in a different location and see if all the files are in the server.

PS5-1 Binary-Tree Re-balancing (60 points)

In PS5-1, you implement the binary-tree re-balancing algorithm presented by Quentin F. Stout and Bette L. Warren in 1986. (http://web.eecs.umich.edu/~qstout/pap/CACM86.pdf)

(1) Download and read the paper.
(2) Implement right rotation function in the BinaryTree class in:

    ~/24783/src/*yourAndrewId* utility/bintree.h
(3) In main.cpp, the user needs to be able to apply right-rotation by R key while the mouse cursor is on the target node.
(4) Implement TreeToVine function in the BinaryTree class.
(5) In main.cpp the user needs to be able to apply tree-to-vine by V key.
(6) Implement Compress and VineToTree function.
(7) In main.cpp the user needs to be able to apply vine-to-tree by T key.

(8) In main.cpp when the user presses the SPACE key, insert a new random number (between 0 and 99) to the binary tree.

If you successfully implement the four member functions and modify main.cpp accordingly, you must be able to re-balance the tree by pressing V and T keys.

PS5-2 AVL-tree (40 points)

In PS5-2, you implement a type of self-balancing binary-tree called AVL-tree. Read the explanation in the lecture note for more details about the algorithm.

(1) In BinaryTree class, add a flag (bool) called autoRebalance. Make it a public member variable.
(2) In the constructor, set autoRebalance to false so that by default auto-rebalancing is off.
(3) Write a protected member function called Rebalance, which takes a node pointer (or can be a handle) and applies the re-balancing algorithm of the AVL-tree. This function must check and re-balance the given node and all the up-stream nodes.
(4) In Insert function, if autoRebalance flag is true apply Rebalance at the appropriate location.
(5) In Delete function, if autoRebalance flag is true apply Rebalance at the appropriate location.
(6) In ps5_2/main.cpp, set autoRebalance flag to true in Initialize function BEFORE ADDING NUMBERS.
(7) In ps5_2/main.cpp when the user presses the SPACE key, insert a new random number (between 0 and 99) to the binary tree.

Test your .CPP and .H files with the compiler server

Your final version code must pass the compiler server. See instructions on the Canvas.

Commit your files to the server.

Also, **check out** your submitted files in **a different location** and make sure all files are submitted, and the file contents are the ones you wanted to be graded.

AVL-tree Performance Competition

We measure time that your AVL-tree class takes for adding 10,000,000 random numbers, and then deletes all even-numbers. Top-3 submissions will get 20 bonus points. Next 2 will get 10 bonus points.

If your code beats my AVL-tree class that I'm using for my research work, you'll get 3% bonus toward the final grade. By the way, my code takes 11.50 sec for insertion, and 1.44 sec for deletion, 12.93 sec in total (Windows 10 on Core i7 3.4GHz compiled in Release mode using Visual C++ 2012). It is not the

fastest code in the world. Some parts are written in a conservative fashion. You have a good chance of beating it!