# CSE331 – Project 4: Binary Search Trees

In this project you will complete an implementation of the binary search tree data structure and use it to store and manipulate data from the Internet Movie Database. The project must be submitted to Handin no later than 11:59 pm Friday February 15, 2012. Email the TA with any questions. Good Luck!

## *Project Description*

You have been given code to implement a program that will load and manipulate the data from the Internet Movie Database. The program accepts three arguments from the command line: (1) the imdb datafile, (2) the number of actors (or -1 to read the entire list) and (3) a file containing the list of commands. The executable accepts the following commands:

1. height – prints out the height of the BST
2. actor-movies – list all of the movies for a given actor, the actor is given as an argument
3. movie-actors – list all the actors for a given movie, the movie is given as an argument
4. delete-actor – delete an actor from the database, the actor is given as an argument
5. delete-movies – delete a movie from the database, the movie is given as an argument

All commands occupy a single line in the file, as do their arguments. An example command file is provided, called test.txt. The code to load the database and handle these commands is already written for you. When the actors and movies are completely loaded, they will contain cross-references to one another. When an actor is deleted, its references are also deleted. When all the actors from a particular movie have been deleted, that movie is also deleted. The reverse is also true for movies and actors.

Your job is to complete the Tree.h file which has been provided. You must provide an efficient implementation of a binary search tree. Most of your functions should have a best case running time of O(log N), however depth and preorder will always take O(N), since they must recurse over every node.

You may not use anything from the STL for this project. You may use IO for debugging, but be sure to remove it from your completed project. You will be marked down for any debugging IO left in your project. As always you may not change the signatures of the public methods, the node class, or the way in which the tree class interacts with the rest of the program. You may add any private member variables or methods to CTree to help you complete the project. You may also assume that any templated types passed to CTree will have overloaded output stream operators (aka “<<”).

## *Programming Notes*

A sample database file is provided in the project directory, this is a subset of the full imdb dataset and it has all of the data necessary to work with the sample command file. These sample files are provided to help you test primary functionality. You should perform much more rigorous testing on your completed projects and make sure to test them on Adriatic! When we grade your project 4's we will be using the full dataset and a much longer command file.

In the Makefile you will notice a variable at the top of the file, CPPFLAGS. This variable contains the flags we want to pass to the compiler. If you are debugging your code, then you would probably want to change the flags to: -g -Wall. I will be compiling your projects with the flags: -g -O3.

In the directory ~cse331/Projects/Examples we have placed a windows program that you can use to explore the behavior of trees, both balanced and unbalanced. The executable is called “VisualTree.exe”.

I recommend that you implement all of your functions in the private space, and then place calls to private methods in the public methods. This will reduce the amount of code you write when overloading the const methods.

## *Project Deliverables*

The following files must be submitted via Handin no later than 11:59 pm Friday February 15, 2012:

- Tree.h – contains your implementation of binary search tree.
- project4.pdf – files containing your answers to the written questions. (Must be a Portable Document Format file)

## *Written Questions*

Submit the answers to these questions in a PDF file, called project4.pdf, along with your source code files. The versions of MS Office and OpenOffice in the CSE labs both support exporting to PDF format. Any images or diagrams must be embedded in your PDF file. They will not be graded if they are separate files. For problems asking for a short paragraph, the paragraph should consist of 4-5 complete and concise sentences.

1. Give a short paragraph to justify your answers to part (a)
    a. Show that the maximum number of nodes that can be stored in a BST is $2^{h+1}-1$.
    b. What are the heights of your movie ($h_m$) and actor ($h_a$) trees? Given the heights $h_m$ and $h_n$, how many nodes can be stored in each tree, how many nodes are actually stored in these trees?
2. Suppose we have the following set of numbers {10, 11, 15, 19, 23, 78, 42, 56, 18, 13, 12, 38, 47}
    a. Determine the order of insertions with this set of numbers that will result in a perfectly balanced BST.
    b. Show the results of a preorder traversal of this tree.
    c. Delete the root, make a diagram of the resulting tree.
3. A full node is a node with two children. Prove that the number of full nodes plus one is equal to the number of leaves in a non-empty binary tree.
4. Suppose we have numbers between 1 and 1000 in a binary search tree and want to search for the number 363. Which of the following sequences could not be in the sequence of nodes examined? Describe the logic you used to determine this in a short paragraph. (CLARIFICATION: for any given sequence (...a,b,c...) node b is the immediate child of a, and the immediate parent of c).
    a. 2, 252, 401, 398, 330, 344, 397, 363
    b. 924, 220, 911, 244, 898, 258, 362, 363
    c. 925, 202, 911, 240, 912, 245, 363
    d. 2, 399, 387, 219, 266, 382, 381, 278, 363
    e. 935, 278, 347, 621, 299, 392, 358, 363
5. We can sort a given set of n numbers by first building a binary search tree containing these numbers, with repeated insertions, and then printing the numbers in an in-order fashion. What are the best case and worst case big-Oh running times for this sorting algorithm? Justify your answers with a short paragraph.