**CS 280 - Programming Assignment Binary Tree**

**Information**

* [Handout](Handout.docx)
* BST interface: [Text](BSTree.h)
* AVL tree interface: [Text](AVLTree.h)
* Sample driver program: [Text](driver-sample.cpp)
  + Output from the driver program.
    - [Output](output-all-nobf.txt)   No balance factor, no counts
    - [Output](output-all-nobf-counts.txt)   No balance factor, with counts
    - [Output](output-all-bf.txt)   Balance factor, no counts (extra credit)
    - [Output](output-all-bf-counts.txt)   Balance factor with counts (extra credit)
    - [Individual output](outputs.zip) for each test, with and without BF and counts. Remember, running tests in different orders will generate a different sequence of random numbers so these won't match the outputs from the links above.
* Sample command lines (for the templated version):
  + Microsoft:

**cl** -Fems.exe driver-sample.cpp ObjectAllocator.cpp PRNG.cpp /EHa /W4 /WX /MTd /Zc:forScope /D\_CRT\_SECURE\_NO\_DEPRECATE

* + GNU:

**g++** -o gnu driver-sample.cpp ObjectAllocator.cpp PRNG.cpp -O -Wextra -Wall -pedantic -Wconversion

* + Borland:

**bcc32** -v -vG -w **-w-8026 -w-8091** -ebor.exe driver-sample.cpp ObjectAllocator.cpp PRNG.cpp

* [FAQ](FAQ.docx) for this assignment.
* [Dummy OA](dummy_oa.zip) Dummy ObjectAllocator to use for this assignment.

**An Example SpellChecker**

* [Spell check](driver-spell.cpp) stress testing. It's a simple demonstration of an efficient spellchecker. This is obviously far superior to the way we did it in CS120 and CS170.

The program will accept 3 command line arguments:

driver-spell [test\_number] [dictionary] [show\_tree]

Example:

driver-spell 2 allwords.txt 1

This will run the second test with the dictionary **allwords.txt** and turn on the displaying of the tree. You can download a few sample dictionaries [here](dictionaries.zip). Here's an example running this:

driver-spell 2 lexicon.txt 1

Output:

====================== TestSpellCheck1 ======================

type: AVLTree, height: 3, nodes: 12

MANGO

GRAPE PINEAPPLE

BANANA LEMON ORANGE STRAWBERRY

APPLE CHERRY KIWI LIME WATERMELON

Value LIME found with 4 compares

Value ORANGE found with 3 compares

Value WATERMELON found with 4 compares

Value FRUIT NOT found with 5 compares

The same thing running test 1 (BSTree instead of AVLTree):

====================== TestSpellCheck1 ======================

type: BSTree, height: 11, nodes: 12

APPLE

BANANA

CHERRY

GRAPE

KIWI

LEMON

LIME

MANGO

ORANGE

PINEAPPLE

STRAWBERRY

WATERMELON

Value LIME found with 7 compares

Value ORANGE found with 9 compares

Value WATERMELON found with 12 compares

Value FRUIT NOT found with 5 compares

**Disclaimer**: Please understand that the code to "draw" the text trees in the output is very simple and crude. Trying to display long strings or many strings will cause the program to crash. I'm not going to try and support printing hundreds or thousands (or more) of strings on the screen. It just won't work! But, for smaller dictionaries, it will try to print them in a readable fashion so you can see and debug your programs. Also, since I'm not really trying to create a working spellchecker, I'm forcing everything into UPPERCASE to keep it simple.

**Pseudocode for indexing**

Note: Near the top of the driver file, there is this line:

**#define** SHOW\_COUNTS

This is what turns on/off the counts when displaying the tree. Remove it and you won't see any counts displayed. If it's defined, you'll see them.

* Since the subscript operator (**operator[]**) is not recursive, you'll need a helper function to do the recursion:

some\_helper(node, index);

where node is the node (initially the root) and index is the position in the tree.

Suppose we're looking for the *i'th* node and we're at some node (the root initially). Assume that *L* is the number of nodes in the left subtree.

* + If node is NULL, return NULL. (Base case)
  + If the left subtree has **more** than *i* nodes (*L > i*), then it's in the left subtree.
    - Recursively call the method with the left node and the index.
  + If the left subtree has **less** than *i* nodes (*L < i*), then it's in the right subtree.
    - Recursively call the method with the right node and *i - L - 1.*
  + Else the left subtree has exactly *i* nodes so it's in the current node.

Looking at the example below, the values represent: Key[count]. Key is the value in the node and count is the number of nodes in the (sub)tree. Because I inserted Keys from 0 to 9, the Key *is* the index as well. This makes following the example below easy. On non-sequential data (random, strings, etc.) this will not necessarily be the case.

5[10]

2[5] 8[4]

0[2] 3[2] 6[2] 9[1]

1[1] 4[1] 7[1]

Suppose we're looking for *tree[3]* (*L* is the number of nodes in the left subtree, *i* is the index):

* + Starting at the root:
  + *L > i* so it's in the left subtree. (Recurse left with node 2[5] and *i*, which is 3)
  + Now, *L < i*, so it's in the right subtree. (Recurse right with node 3[2] and *i - L - 1*, which is 0)
  + Now, *L == i*, so the value is in the current node, 3[2].
  + Return the node.

Suppose we're looking for *tree[8]* (*L* is the number of nodes in the left subtree, *i* is the index):

* + Starting at the root:
  + *L < i* so it's in the right subtree. (Recurse right with node 8[4] and *i - L - 1*, which is 2)
  + Now, *L == i*, so it's in the current node, 8[4].
  + Return the node.

Suppose we're looking for *tree[7]* (*L* is the number of nodes in the left subtree, *i* is the index):

* + Starting at the root:
  + *L < i* so it's in the right subtree. (Recurse right with node 8[4] and *i - L - 1*, which is 1.)
  + Now, *L > i*, so it's in the left subtree. (Recurse left with node 6[2] and *i*, which is 1.)
  + Now, *L < i*, so it's in the right subtree. (Recurse right with node 7[1] and *i - L - 1*, which is 0)
  + Now, *L == i*, so the value is in the current node, 7[1].
  + Return the node.

Try a couple more on your own to convince yourself that this works.

Notice that you don't necessarily have to do any checking of the subscript. For example, if I ask for *tree[12]* (non-existent), it will eventually recurse on an empty child (NULL). Be sure to return NULL from the method, indicating that nothing was found. This is the non-recursive case (base case) in the algorithm. Of course, as an optimization, you could just check the value of the index and if it's out of range, just return NULL. We aren't throwing any exceptions for an out-of-bounds index.

Finally, realize that this doesn't have to be recursive as you are only walking down the tree.

**Other Techniques**

1. The public interface will not be able to do the recursion. You'll need to do things like this with a private method:

**template** <**typename** T>

**void** BSTree<T>::insert(**const** T& value) **throw**(BSTException)

{

*// Call a private recursive method*

insert\_item(root\_, value);

}

1. Using the stack-based method demonstrated in class, you'd have something like this:

**template** <**typename** T>

**void** AVLTree<T>::insert(**const** T& value) **throw**(BSTException)

{

*// Create a stack to use with the private recursive method*

std::stack<**typename** BSTree<T>::BinTree \*> nodes;

insert\_item(**this**->get\_root(), value, nodes);

}

Notice the **typename** keyword and the *non-redundant* (read: required) use of **this** above. You could do this as well:

insert\_item(BSTree<T>::get\_root(), value, nodes);

See [Name lookup, templates, and accessing members of base classes](http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/4/html/Using_the_GNU_Compiler_Collection/c---misunderstandings.html#NAME-LOOKUP) for a refresher on this.

1. Here's how you deal with the exception if there is no more memory available. Since you're using the ObjectAllocator, you won't be calling **new** for any nodes. (You won't be calling **delete** either.) However, you will be using placement new to construct the data.

*// Placement* ***new*** *won't work with the overloading of* ***new*** *for leak checking using Microsoft's API.*

*// It will actually detect the leak, but it won't give you filename and line number.*

**try**

{

Node \*mem = **reinterpret\_cast**<BinTreeNode \*>(objAllocator\_->Allocate()); *// Allocate memory for the object.*

node = **new** (mem) BinTreeNode(value); *// Construct the object in the memory.*

}

**catch** (**const** OAException &e)

{

**throw**(BSTException(BSTException::E\_NO\_MEMORY, e.what()));

}

1. Don't forget to call the destructor explicitly on the node when you remove it from the list. Since the class is templated, you don't know if any of the data needs to be destroyed via its destructor. For example:

some\_node->~BinTreeNode(); *// Call the object's destructor.*

objAllocator\_->Free(some\_node); *// Free the object's memory.*

1. You'll need to implement this function as appropriate:

**template** <**typename** T>

**bool** AVLTree<T>::ImplementedBalanceFactor(**void**) **const**

{

*// Change to true if you used the efficient method and want extra credit*

**return** **false**;

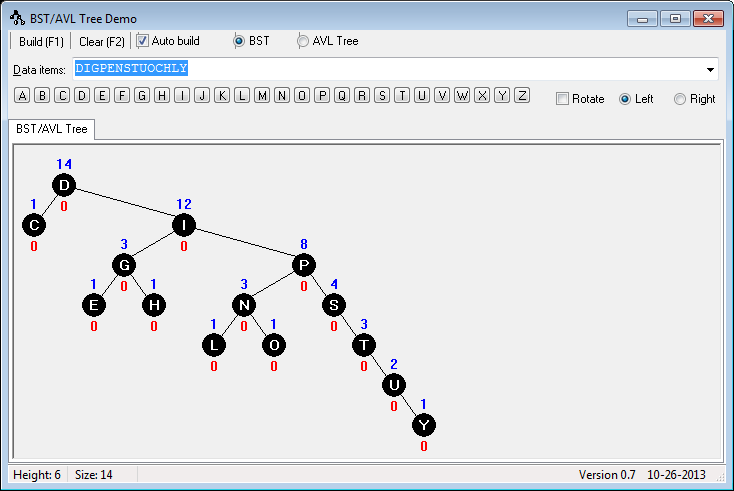
}

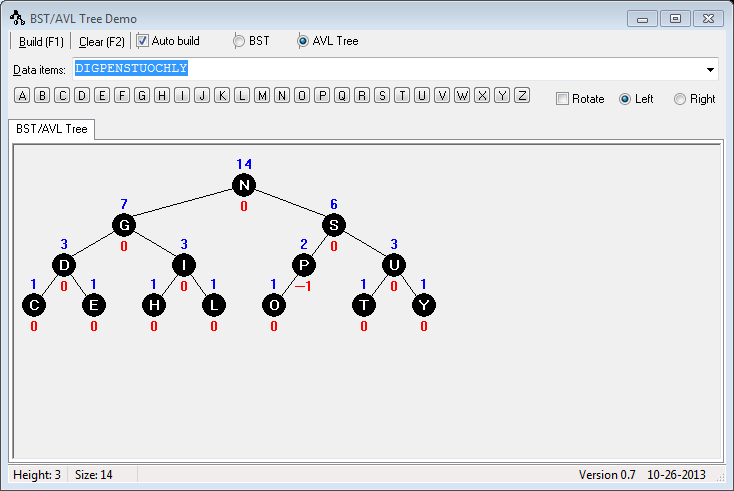
1. Additional information on [AVL trees](https://faculty.digipen.edu/~mmead/www/Courses/CS280/avl-2.0.1/AVL-Trees.html) and [balance factors](https://faculty.digipen.edu/~mmead/www/Courses/CS280/avl-2.0.1/Step-3-in-AVL-Insertion.html) for those wishing to do the extra credit.

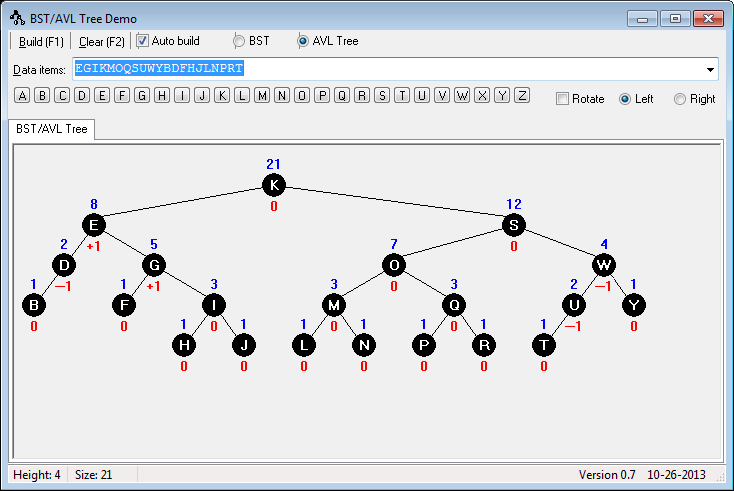
**GUI Driver**

No installation necessary. Just copy to your hard drive and run it. It's a work in progress, so it may not seem intuitive yet. You can just type in some letters in the box or click on a button (letter) to add it to the tree. If the letter exists, clicking on a button will remove it. If you want to rotate about the node, select the Rotate checkbox. It's still pretty old. One of these days I'll update it (like the one for Red-Black trees.)

[Download](BSTAVLTrees.exe) **New and Improved!** with counts and balance factors. The node count is in BLUE, above the node, and the balance factor is in RED, below the node. The balance factor only works with AVL trees, not BSTs.







[Download](https://distance.sg.digipen.edu/file.php/1659/Assignment4/BSTAVLTrees-original.exe) The Original™ (without counts and balance factors).

