CMSC 341 Project 3

Base Data Structure

AVL tree

Augmented Data Structure

AVLForest = vectors and/or maps + vector of templated AVL trees

Implementation Overview

AVL tree
Templates
Operator overloading
File I/O
STL Vectors and/or STL Maps

Project Outline

A Binary Search Tree (BST) is a binary tree which maintains an ordering between nodes. While the height of a BST is O(log(n)) on average (for n elements), this is not guaranteed. In the worst case, the elements are inserted in ascending or descending order. In that case, the insert/search/delete operations have complexity O(n) (and that makes us sad).

An Adelson-Velsky Landis (AVL) tree is an extension of a BST, which enforces that the heights of the two child subtrees of any node differ by at most one. If this rule is violated, the tree is rebalanced and the AVL tree property is restored. Rebalancing is implemented using tree rotations. In doing so, the worst case complexities for insert/search/delete operations become O(log(n)) (and that makes us happy!). Some useful explanations of AVL trees can be found here: [Text] [Video] [Simulation]. Also remember that a double rotation is just a combination of two single rotations. For a given sequence of insertions, there is a fixed sequence of rotations and the resultant tree is therefore unique. Additionally, one rotation (whether single or double) is all that is needed to rebalance after an insertion causes an imbalance. Rebalancing is therefore O(1) time.

In this project, we insert <character, number> pairs into an AVL tree, **sorted by the number**. If the AVL tree is implemented correctly, the corresponding characters will get sorted in a particular order. A traversal of the tree will then spell out a word (printing an AVL tree / node in an AVL tree may be overloaded to do so).

In a similar case, the AVL tree may correspond to <word, number> pairs, resulting in an AVL tree corresponding to a sentence. The AVL trees are therefore to be templated for use with inputs as <T, number>. The numbers can further be evaluated as integers or floats, which would also allow the “number” field to be templated. So we now have the AVL tree templated as <T_1, T_2>. For simplification, there will be consistency of data types among all the insertions for a given AVL tree.

The traversal requirement for an AVL tree may be in-order, pre-order, post-order or level-order based on specification for the AVL tree. Printing the AVL tree with the specified order then allows for a form of “encrypted” message passing mechanism (with a poor level of security, but it’s something ¯_(ツ)_/¯). For consistency of print order, the following enumerator is mandated to be used (included in given code):

enum PrintOrder{IN, PRE, POST, LEVEL};

Data will be read from an input stream (file), which will correspond to an unknown number of AVL trees to be created and traversed, each with their own <T_1, T_2> pairs and print order requirements. Similar to a real-time data source, all inputs for a given tree may not occur in a single block in the input file; inputs across AVL trees may be interleaved in the input file (see example).

Input Description

An input file will contain the first entry for any tree of the format:
<AVL tree ID,order of printing(enum),data type (enum),number type (enum)>

All following entries for the tree will then be present in the format:
<AVL tree ID,T_1,T_2>

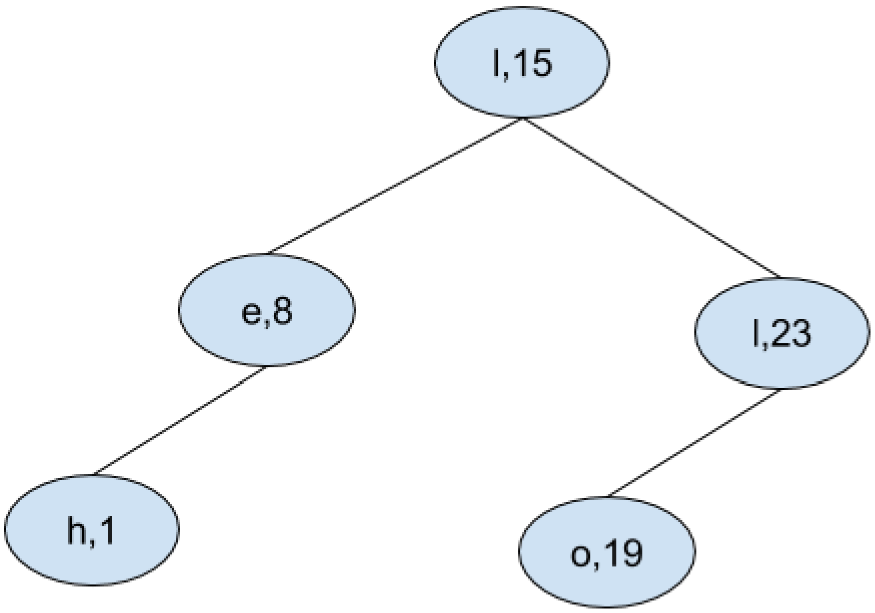
A tree ID is a unique positive integer identifier for the tree. For a given AVL tree, all input entries will have consistent AVL tree ID and print order. They will also be consistent on data type instantiations of T_1 and T_2 (for example, all entries will use strings and integers). All numerical values used (integers and/or floats) will be positive. For simplicity, they will also be unique.

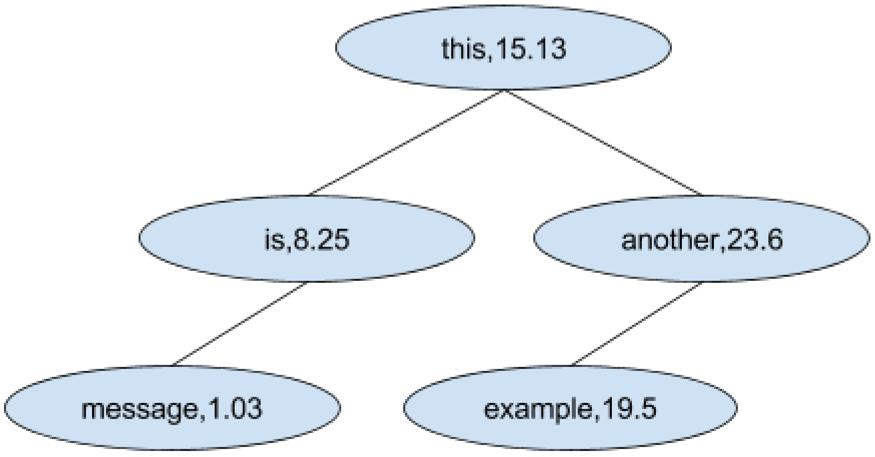
Example input file:
```
1,0,0,0
1,e,8
25,3,1,1
25,is,8.25
1,l,15
25,this,15.13
25,another, 23.6
1,o,23
25,message,1.03
1,h,1
```

1,l,19
25,3,example,19.5

Generated AVL Trees

The two AVL trees corresponding to the input are as follows.





Output description (part 1)

The output will be the output of one AVL tree per line, when printing. This corresponds to a sentence (for character nodes) or a set of sentences (for word nodes), per line. Note that capitalization is preserved.

Example output:

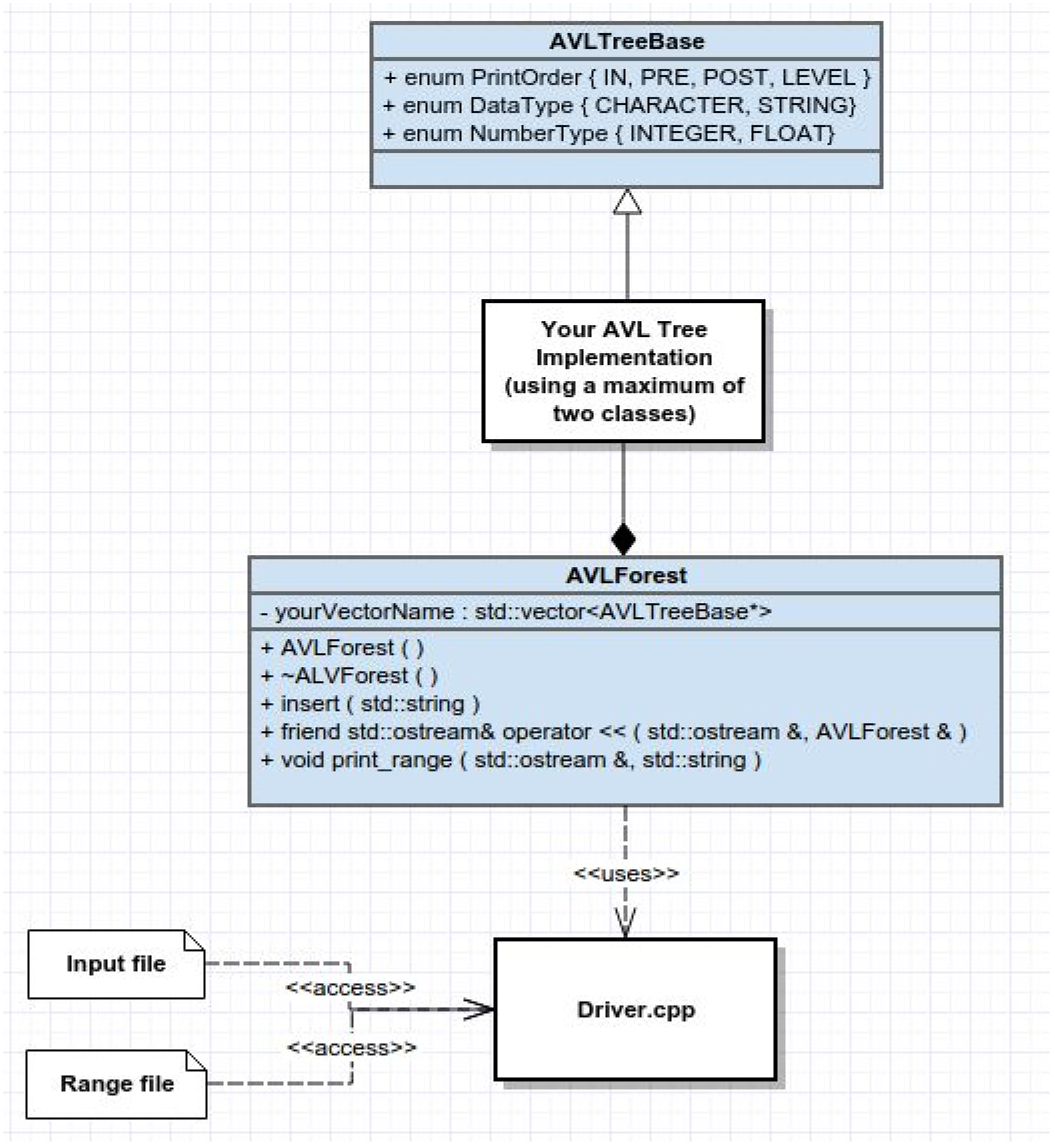
h e l l o
this is another message example

Complexity Requirements

All insert operations for an AVL tree should be O(log(n))
print_range (explained later in this text) should be O(m + log (n))
All other print operations (based on PrintOrder) should be O(n)

n = number of nodes in the tree
m = number of elements in the specified range

Code Details Provided


AVLTreeBase
+ enum PrintOrder { IN, PRE, POST, LEVEL }
+ enum DataType { CHARACTER, STRING}
+ enum NumberType { INTEGER, FLOAT}
Your AVL Tree Implementation (using a maximum of two classes)
AVLForest
- yourVectorName : std::vector<AVLTreeBase*>
+ AVLForest ( )
+ ~ALVForest ( )
+ insert (std::string)
+ friend std::ostream& operator << (std::ostream &, AVLForest &)
+ void print_range (std::ostream &, std::string)
<<uses>>
Input file
<<access>>
Driver.cpp
<<access>>
Range file

Driver.cpp: **This file is not to be changed**. Expect your implementation to be used in this manner.

AVLTreeBase.h: **This file is not to be changed**. In order to create a single vector of AVL trees which may have varied template instantiations, it is mandated that the templated AVL tree implementation be derived from the AVLTreeBase class. An AVL forest can then instantiate a vector of AVLTreeBase pointers (since AVL trees are created at run time as and when required), without template specification. This base class only contains a definition of the enumerators used for tree print order and specification of data types used by a tree.

AVLForest.h/AVLForest.cpp: Minimal skeletons of the forest implementation. You may add data members/member functions as felt necessary. Because we must provide template instantiations when dealing with AVL tree objects, AVLTreeBase related objects must be converted to templated AVL tree objects when using their methods. This can safely be done using static_cast and has been exemplified in the code skeletons. Note that “yourVectorName” and is a placeholder. Please replace this with something more informative. The << operator for a forest must print all trees in the vector (in sequence in which they were first encountered in the input file).

Code Details Not Provided

There is not an extensive UML specification provided for the implementation of an AVL Forest and no specification for AVL tree implementation (apart from the mandated inheritance). The reason for this is to encourage you to think about design based on what you have learned so far. The AVL tree implementation is **limited to a maximum of two classes**.

The AVL tree must implement all four print orders specified in the PrintOrder enum. Additionally, the AVL tree must also implement a function named **print_range** (must be called this). This function accepts three arguments: output stream, numerical upper bound and numerical lower bound. print_range(out, x,y) prints (to out) the items in the AVL tree that have key values between x and y (**inclusive**) in increasing order (by key value). The running time of the function must be O(m + log n) where n is the number of items in the tree and m is the number of items between x and y. It is therefore not feasible to just do an inorder traversal and not print the items that fall outside the range (because that would take O(n) time even when x == y). A hint for this part is to think about iterators in data structures. Note that the print_range function implemented for the AVL tree is different from the print_range function implemented for the AVL forest. Intuitively, the latter will make use of for former.

The limits for various trees will be specified in a separate (range) file in the format
<AVL tree ID,lower limit, upper limit>

The IDs in this file will be consistent with the IDs used by input.txt. The upper and lower limit values will be consistent in data type with the numbers in the tree.
Example range file:

```
1,8,16
25,8.25,16.52
```

Output description (part 2)

The output to be generated by using print_range is as follows (with the example range file and trees).

Example output:

```
e l
is this
```

Command Line Arguments

There are two command line arguments provided to the “run” target. Those are:

INPUT: the name of the input file for creation and population of trees
RANGE: the name of the range file for using print_range on specified trees with specified bounds

E.g.: `make run INPUT=input.txt RANGE=range.txt`

Memory Leaks

The code should not exhibit memory leaks (or segmentation faults).

Exception Handling

You are not required to throw exceptions. Graceful termination (no segmentation fault) in unexpected cases (such as the range file specifying a tree that does not exist) is sufficient.

What to Submit

Read the course project submission procedures. Submission closes by script immediately after 9pm. Submit well before the 8:59pm deadline, because 9:00:01 might already be late (the script takes only a few seconds to run).

You should copy over all of your code under the src directory. You must also supply a Makefile. Do NOT submit your own test data files. Any unnecessary files submitted will be considered for a deduction.

Make sure that your code is in the ~/cs341proj/proj3/src directory and not in any other subdirectory of ~/cs341proj/proj3/. In particular, the following Unix commands should work.

```
cd ~/cs341proj/proj3/src
make
make run INPUT=<input file name> RANGE=<range file name>
make clean
```

The command “make run” should simply run the project that compiled successfully.

Don't forget the project submission requirements shown online! One hint, after you submit, if you type:

```
ls ~/cs341proj/proj3/
```

and you see a bunch of .cpp and .h files, this is WRONG. You should see:

```
src
```

instead. The C++ programs must be in directories under the src directory. Your submissions will be compiled by a script. The script will go to your proj3 directory and run your makefile. This is required. You will be severely penalized if you do not follow the submission instructions.

Due date

Please check Blackboard.