CSE 373 SPRING 2016: HW3
Binary Search Trees:
Keyword Search

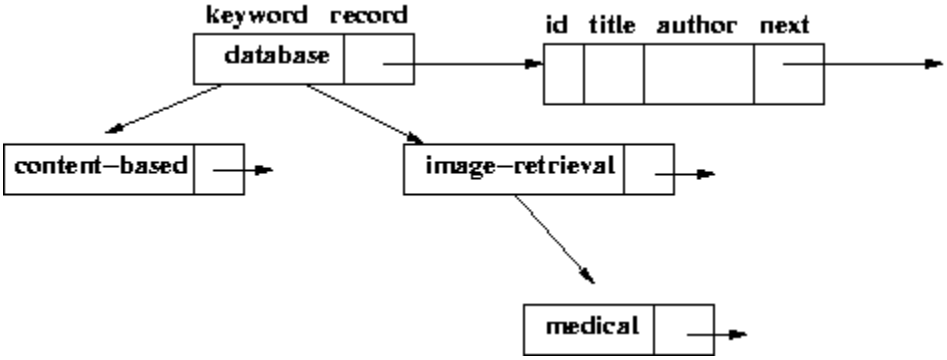
Assigned: 4/13/2016, **Due:** 4/22/2016

Background

Information retrieval systems allow users to enter keywords and retrieve articles that have those keywords associated with them. For example, once a student named Yi Li wrote a paper called, "Object Class Recognition using Images of Abstract Regions," and included the following keywords: 'object recognition', 'abstract regions', 'mixture models', and 'EM algorithm'. If someone does a search for all articles about the EM algorithm, this paper (and many others) will be retrieved.

Assignment

You are to implement a binary search tree and use it to store and retrieve articles. The tree will be sorted by keyword, and each node will contain an unordered linked list of Record objects which contain information about each article that corresponds to that keyword. This image shows the idea:



Getting started: The necessary files are available at `http://www.cs.washington.edu/education/courses/cse373/16sp/homework/hw03/hw03.zip`. Included in this archive are:

- **datafile.txt:** A text file which contains records to be read into the data structure.
- **Record.java:** The "Record" class will be the objects stored in the value for each keyword in the tree. This class also has a "next" pointer which provides the structure for the linked list. Objects of this type will be the value of each node in your search tree. This code should not be modified.

- **Test.java**: This code performs reading from the data file as well as allowing test operations of your binary search tree. Performing changes to this file can be done to test particular cases, but this is for your benefit, since it will not be collected. The code provided will print the contents of the tree in inorder, which is alphabetical order. At each node of the tree, it will print the key word and then the titles of all the records in the list that you have created at that node. The test code also performs a few deletions and checks the result to ensure that delete() works correctly.
- **BST.java** This file contains the basic shell for the data structure we ask you to implement in this assignment. All functions that you are asked to implement are marked with a //TODO comment and are listed below with their expected operation. **You will need private helper functions to implement some of these recursively.**
  - Node constructor: This function should initialize a record with keyword 'k'. It does not require the other fields to be set because every Node construction will be updated either by directly modifying the children or by performing an update() to add a record to its linked list.
  - Node update(Record r): This function should add the Record r to the linked list for a particular keyword. You should add new Records to the front of the list.
  - insert(String keyword, FileData fd): This function includes code that turns the FileData fd into a Record recordToAdd. The function should insert recordToAdd to the node of keyword. If keyword is not in the tree, it should create a node.
  - contains(String keyword): This function should return true if the keyword is in the tree.
  - get_records(String keyword): This function returns the linked list of Records associated with a given String keyword.
  - delete(String keyword): This function removes the node associated with the input string keyword. If no such node exists, the code should do nothing.

Grading: 35 points

- Node constructor: 3 points
- Node update(Record r): 3 points
- insert(String keyword, FileData fd): 6 points
- contains(String keyword): 5 points
- get_records(String keyword): 6 points
- delete(String keyword): 7 points

Additionally, there will be 5 points awarded for readability of code and comments for a total of 35 points.

Submission for this assignment only requires BST.java to be sent to the class dropbox. The data file provided will be used for grading, so feel free to edit Test.java in any way that will make you confident that your binary search tree performs as a binary search tree is supposed to, that is:

1. Each node satisfies the binary search tree property that its key is greater than the key of its left child and less than the key of its right child.

2. Insertions and deletions are done correctly and do not violate the binary search tree property.

3. Empty tree situations are handled properly.

4. All titles for a given key word are placed in the list at the node for that key word; they should be inserted at the BEGINNING of the list.

Extra Credit: 10 points

Copy BST.java and change the name of the new file into AVL.java. The class name (BST) and method definitions (insert, contains, get_records etc.) should not change so that Test.java can use the new class without any modification. Then, convert the BST implementation to a self-balancing AVL tree (AVL trees are balanced BSTs). This will require coding rotation methods that will keep the tree balanced if an insert will result in an unbalanced tree. Your code does not need to incorporate deletions.

A description of the AVL tree and its rotations begins on page 123 of the Weiss textbook. Completing this extra credit will require adding several attributes to the Node and BST tree provided. If you plan to complete the extra credit, **please submit the AVL section of your submission as AVL.java in addition to the BST.java** required in the regular portion of the homework.