Assignment 5
Trees and Binary Search Trees

Release Time	Due Date
November 18, 2016	**Part1 due ~~11/28/2016~~; Part2 due 12/1/2016**
	part1 extended to 11/29/2016

Objectives

- To understand trees data structures
- To practice programming using binary search trees
- Practice developing high-performance solutions.
- Analyze what are the advantages and the disadvantages about various techniques to manipulate trees.

Problem Specification

In this assignment we will get practice in manipulating trees.

PART 1 (due 11/28/2016):

1. Derive the relationships among height h, total number of nodes n, number of leaves L, and the number of edges m in a complete quad tree T. In particular, given h, what is n? Given n, what is h? Given m, what is h? Given m, what is n? Given L, what is h? Given h, what is L? Show all your work otherwise no credit.

2. Repeat question 1 if T is a full quad-tree.

3. Describe an implicit representation of a quad-tree T using arrays. Give the declaration of the array that stores the data of the nodes of T. Assume that the data consists of three fields: ssid, name, phoneNumber. Given a node v of T at index i, what are the indices of the four children of v and the parent of v?

4. Let T be a (unbalanced) binary search tree:

    a. Insert the sequence 65, 70, 60, 72, 87, 40, 35, 90, 75, 63, 68, 69, 61, 20, 25, 28, 37 (of integer keys) into T. Show the tree after each insert operation.

    b. Show the output of Preorder, Inorder and Postorder traversals of T after last insertion (i.e., after key 37 is inserted).

    c. Delete the sequence 28, 72, 65, 35, 63, 87, 70 from T. Show T after each deletion. Note that to be consistent in deletion, follow the same algorithm when you have to "borrow" a key to fill a hole.

PART 2 (due 12/01/2016):

5. Design, develop and implement an object-oriented application to build an unbalanced binary search tree T using an array based implicit representation of T and starting with an empty tree T. The object representing T should support at least the following operations: *insert(mydata x), delete (mydata x), search (mydata x), preorderTraversal( ), inorderTraversal( ), and postorderTraversal( )*. mydata is a record containing following three fields: stuName (a string of at most 15 characters, first and last name separated by a colon ':')), courseNumber(an integer), grade (a char). Search key in T is based on stuName. Use recursion to implement the three traversals. Use dynamic allocation of array elements whenever possible (hint: c# and Java users can use arraylist).

One preliminary object-oriented design would be to define at least two classes: one mydata and another called binaryStreeImplicit with appropriate data members and methods, a skeleton is shown below. (Feel free to choose any other object oriented or top-design or make it better.)

```
Class mydata { String stuName; int courseNumber; char grade;
                        … // properties, constructors and methods }
Class binaryStreeImplicit {
         mydata[ ] tree; // holds all the nodes of the binary search tree
          int treeSize; // number of nodes in the binary search tree
          int lastIndexUsed; // the tree may not be full,
                                    // so specifies the array bound
          … // properties and constructors
         // some methods that may be helpful
          int root(); //returns the index of the root of the tree or -1 if
tree is empty
          int leftchild(int i); int rightchild(int i); int parent(int i);
                  // return the indices of the children and
                  // parent of a node at index i
           void inorderTraversal(); void preorderTraversal();
void postorderTraversal();
                //obvious meanings;  prints data of nodes one line at a
time; prints only the nodes where actual data exists (i.e., no holes)
          int insert (mydata x); // inserts x into the tree; returns the
index of the array where inserted, or -1 if unsuccessful
           int delete (mydata x); // deletes x from the tree – note this
may create "hole" in the array storing the tree nodes but BST is
maintained; returns -1 if x doesn't exist, otherwise the index where x was
          int search (mydata x); // returns the index where x exists,
otherwise -1
           … // add any other private / public methods that may help
manipulation of BST
              }
```

Sample input demo text file. Format of the demo file: a number of lines, each line containing an operation to be executed on the BST. For example,

```
Insert: John:Doe, 3310, A
```

```
Insert: Jane:Dane, 1120, B
Delete: June:Doe
Delete: John:Doe
Insert: Aj:Gup, 3310, B
Insert: Foo:Done, 2240, C
Inorder
Preorder
Insert: Aj:Gupt, 3310, D
Postorder
Search: John:Doe
Search: Jane:Dane
Inorder
…
```

Design Requirements

Code Documentation

For this assignment, you must include documentation for your code as generated by JavaDoc. You should have JavaDoc comments for every class, constructor, and method. By default, JavaDoc should output html documentation to a subfolder within your project (/dist/javadoc). Make sure this folder is included when you zip your files for submission. You do not need to submit a hard copy of this documentation.

Hint: http://stackoverflow.com/questions/4468669/how-to-generate-javadoc-html-in-eclipse

Coding Standards

You must adhere to all conventions in the CS 3310 Java coding standard. This includes the use of white spaces for readability and the use of comments to explain the meaning of various methods and attributes. Be sure to follow the conventions for naming classes, variables, method parameters and methods.

Testing

Make sure you test your application with several different values, to make sure it works.

Assignment Submission

- Generate a .zip file that contains all your files, including:
  - Source code files
  - Including any input or output files
- Javadocs
- A report containing analysis of your implementation (theoretical as well as empirical)