Seoul National University

Data Structure

Fall 2020, U Kang

Programming Assignment 2: Binary Trees (Chapter 5)

Due: October 26, 23:59 pm, submit at eTL

Reminders

- The points of this homework add up to 100.
- Like all homeworks, this has to be done individually.
- Lead T.A.: Seungcheol Park (ant6si@snu.ac.kr)
- Write a program in Java **(JDK 14)**.
- **Do not use** Java Collection Framework and the third-party implementation from the Internet.

1. How to submit the programming assignment

1) Create a ***JAR*** file including 'src' folder that contains your sources files. (Refer to '1 – Introduction.pptx' in the first lab session)
    - We will run your ***Main*** class in the JAR file to grade your programming assignments. Before submitting the JAR file, please check if your Main class in the JAR file works correctly.
    - You **MUST** obey the I/O specification of the programming assignment, and rules for the submission of the programming assignment.
    - Before submitting, check if your JAR file runs properly in your terminal with the following commands:

      "`java -classpath PA_##_(StudentID).jar com.dmlab.Main`".

      (e.g.java -classpath PA_02_2020-12345.jar com.dmlab.Main)

2) Submit the jar file to the eTL (http://etl.snu.ac.kr/) .

- **NOTE:** you must import the "BookSearch_skeleton" directory when you start this assignment. Please check if you can read the "sample_input.txt" without any modification.

- **NOTE:** if you want to check your jar file working, you have to make jar file and the input file located in the same directory (i.e. make the relative path of the input file to be "sample_input.txt")

2. How to grade your programming assignment

1) We made a grading machine to automatically grade your programming assignment. The machine will run your program and compare answers and outputs that your program generates for given inputs. If your program cannot generate correct answers for an input file, it will not give you the point corresponding to the input. Our machine will consider the following scenarios:
   - (***Accept***) When your program generates exact outputs for an input file, the machine will give you the point of the input.
   - (***Wrong Answer***) When your program runs normally, but generates incorrect outputs for an input file, including typos, the machine will not give you the point of the input.
   - (***Run Error***) When your program does not run, or is terminated suddenly for some reasons, the machine will not give you the point of an input file because it cannot generate any outputs.
   - (***Time Limit***) When your program runs over a predefined execution time for an input file, our machine will stop your program, and it will not give you the point of the input. The time limit of the execution is ***5 seconds***.

2) We will generate 10 input files and assign 10 points for each input file. For example, if your program gets 9 accepts, and 1 wrong answer by the machine, the total point will be 90 points. <u>Hence, before submitting your programming assignment, please be sure that your program makes correct answers in reasonable time for any input case.</u>

3. Problem

Mr. John works as a librarian at a library. His main work is to manage numerous books in the library. When new books are purchased by the library, he needs to put those books on a shelf for customers. Also, books can be discarded if the books are damaged, or nobody has borrowed those books for a long time. Another service is to find books on shelves when a customer asks him where the books are.

To carry out his duties efficiently, he wrote down the information such as the name and the location of a book on a paper. When a customer requires him to find a book, he first searches the book's name and location in the list, goes to the location, and picks up the book. Of course, whenever new books are added, or old books are removed, he updated the list by hand.

However, he is suffering from maintaining the paper list and searching books based on the list as the number of books increases. Hence, he decided to hire you to resolve his problem using a computer. Your main mission is to help him by constructing a search system, which is called *BookSearch* supporting tasks related to books in the library. The basic requirements of *BookSearch* are as follows:

- In *BookSearch*, the information of a book is stored in forms of a key-value pair. The key is the book name, and the value is the location of the book.
- For a book, Mr. John needs to be able to find the location of the book using *BookSearch* with the book name.
- Mr. John needs to be able to add or delete the information of books using *BookSearch*.
- If Mr. John tries to search or delete a book, and there is no book with the name, *BookSearch* should print the message "*BookSearch cannot find the book*".
- The operations of *BookSearch* should be fast to cover a lot of books in the library.

He also requested additional operations as follows:

- *Print book list*: *BookSearch* should list all book names in lexicographical order. If there are no books, BookSearch should print the message “BookSearch does not have any book”.

- *Order search*: *BookSearch* needs to be able to find the given order of the book in the library in lexicographical order, and vice versa. If there is no book with the given order (or name), *BookSearch* should print the message “*BookSearch cannot find the book*”.

- *Validation check*: BookSearch needs to be able to check itself whether it has valid binary search tree structure or not. We determine a binary search tree is valid if each internal node stores a key greater than in the node’s left subtree and less than those in its right subtree when we considering the book’s name in lexicographical order. If there is no book in the tree, *BookSearch* should print the message “BookSearch does not have any book”.

- *Efficiency check*: BookSearch needs to be able to compute ‘naïve tree efficiency η’ of the binary search tree when we call efficiency check function. We define η using following equation.
$$\eta = \frac{n}{2^h - 1}$$
where h and n represents the height of the binary search tree and number of books included in the BookSearch respectively. If η is lower than 0.1, BookSearch should print “BookSearch needs to be reconstructed”. Otherwise BookSearch should print “BookSearch does not need to be reconstructed”. If there is no book in the tree, *BookSearch* should print the message “BookSearch does not have any book”.

4. Specification

Here are several assumptions for clarity.

- In the library, the book names are distinct. There is no duplicate of a book name.

- All book names are in lower case. Also, the book names do not have any white spaces.
- BookSearch should be based on Binary Search Tree (BST).
- Of course, you need to consider that the library contains a lot of books.
- Implement the function for “Order search” using method overloading. The “Order search” uses 1-based numbering (i.e. start with 1, not 0).
- We define a height of a single node tree as one, i.e. the tree which only have a root node has height one.
- There is an additional member variable “size” in the BinaryNode, which represents the size of the subtree (see the result of the print_tree in the sample output). Please carefully manage this variable when you insert or delete new items.

Make “BookSearch” class supporting those requirements. To do that, you need to fill in the “BookSearch.java” and the “BinarySearchTree.java” of “BookSearch” java project. The following is the list of functions you should fill in:

Java file	Function

BookSearch.java	`public void add(String name, String position)`
	
	`public String remove(String name)`
	
	`public String get(String name)`
	
	`public int size()`
	
	`public void printBookList()`
	
	`public String orderSearch(int order )`
	
	`public String orderSearch(String name )`

BinarySearchTree.java	`public void insert(Key key, E value)`
	
	`private BinaryNode<Key, E> insertHelp(BinaryNode<Key, E>`

	`rt, Key key, E value)`
	
	`public E remove(Key key)`
	
	`private BinaryNode<Key, E> removeHelp(BinaryNode<Key, E> rt, Key key)`
	
	`private BinaryNode<Key, E> getMin(BinaryNode<Key, E> rt)`
	
	`private BinaryNode<Key, E> deleteMin(BinaryNode<Key, E> rt)`
	
	`private E findHelp(BinaryNode<Key, E> rt, Key key)`
	
	`private void printBookList()`
	
	`private int printBookListHelper(BinaryNode<Key, E> rt)`
	
	`public Key orderSearch(int order)`
	
	`public Key orderSearchHelper(BinaryNode<Key, E> rt, int order)`
	
	`public int orderSearch(Key key)`
	
	`public int orderSearchHelper(BinaryNode<Key, E> rt, Key key)`
	
	`public int height()`
	
	`public int heightHelper(BinaryNode<Key, E> rt)`
	
	`public boolean validationCheck()`
	
	`public boolean validationCheckHelper(BinaryNode<Key, E> rt)`

Besides these functions, you can freely add new member variables and new memb er functions in BinaryNode.java and BinarySearchTree.java.

To give you an intuitive understanding of BST, we made a class “TreePrinter” that c an print a tree, given the root of the tree. To use this function, you only need to ty pe “print_tree” as the input command in the “sample_input.txt”. Then, run the pr ogram, a “TreePrinter” object will print the current tree. Each node in the tree is a book. To make the tree clear, every node in the tree will be printed as an acronym for the book title and the size of the subtree, for example, the node of “**d**eep_**l**ear ning” with subtree size **3** will be shortened as “**dl3**”.

Before starting programming, we highly recommend you to carefully read Main.java, BinaryNode.java, BinarySearchTree.java and BookSearch.java because they contain many important information you need.

5. Specification of I/O

We will provide the Main.java file that already implemented to fit the I/O format of this assignment. **You should not modify Main.java file and must not add additional prints to the stdout**. We will collect the prints of stdout from your submitted jar and compare with the current answer automatically. Only the exact match will give you a score.

6. Sample Input and Output

- **NOTE**: “Order search” uses 1-based numbering (i.e. start with 1, not 0).

Sample Input:

```
print_all
get history
add gazza_ssanai A4-123
efficiency_check
validation_check
add purity_of_dancer C2-112
add happy_solo B5-331
add uncle_chan C1-100
add fancy_programmer C1-107
print_tree
efficiency_check
validation_check
remove history
size
print_all
add wolverine_vs_superman H1-1
add wolverine_vs_batman H1-2
add wolverine_vs_ironman H1-3
add wolverine_vs_hulk H1-4
add wolverine_vs_spider_man H1-5
order_search uncle_chan
order_search 5
order_search the_dungeon_and_dragon
remove how_to_make_a_girl_friend
remove fancy_programmer
get uncle_chan
efficiency_check
validation_check
```

Sample Output:

```
BookSearch does not have any book
BookSearch cannot find the book
ADD:    gazza_ssanai A4-123
BookSearch does not need to be reconstructed
BookSearch is valid
ADD:    purity_of_dancer C2-112
ADD:    happy_solo B5-331
ADD:    uncle_chan C1-100
ADD:    fancy_programmer C1-107
PRINT_TREE:
  gs5
 / \
/   \
fp1   pod3
  / \
  hs1 uc1

BookSearch does not need to be reconstructed
BookSearch is valid
BookSearch cannot find the book
SIZE:   5
BOOK:   fancy_programmer
BOOK:   gazza_ssanai
BOOK:   happy_solo
BOOK:   purity_of_dancer
BOOK:   uncle_chan
ADD:    wolverine_vs_superman H1-1
ADD:    wolverine_vs_batman H1-2
ADD:    wolverine_vs_ironman H1-3
ADD:    wolverine_vs_hulk H1-4
ADD:    wolverine_vs_spider_man H1-5
ORDER:  5
ORDER:  uncle_chan
BookSearch cannot find the book
BookSearch cannot find the book
REMOVE:         fancy_programmer C1-107
GET:    uncle_chan C1-100
BookSearch needs to be reconstructed
BookSearch is valid
```