Seoul National University

Data Structure

Fall 2015, Kang

Programming Assignment 2 : Binary Tree (Chapter 5)

Due: Oct. 27, 11:00 am, submit at class

Reminders

- The points of this homework add up to 100.
- Like all homeworks, this has to be done individually.
- Lead TA: Jinhong Jung (montecast9@gmail.com)
- Write a program in Java.
- Do not use Java Collection Framework

1. How to submit the programming assignment

1) Create a ***JAR*** file including 'src' folder that contains your sources files, but without 'release' folder. (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 in a terminal.
   - You **MUST** obey the I/O specification of the programming assignment, and rules for the submission of the programming assignment.

2) Compress the JAR file with a readme file and the project folder. If you don't use Eclipse, include source files into the compressed file. The readme file needs to include your student id, name, how to execute your program, and any other information TA needs to know.
   - The format of the JAR file is "*PA_##_(StudentID).jar*" where '##' is the programming assignment ID, and *(StudentID)* is your student ID.
     - ex) PA_01_2015-12345.jar
   - All documents should be written in English.

3) The name of the compressed file (zip file) should be '*PA_##_(StudentID).zip*' where '##' is the programming assignment ID, and (StudentID) is your student ID.
   - ex) PA_01_2015-12345.zip
     - *01: the first programming assignment*
     - *2015-12345: your student ID*

4) Submit the compress file to the eTL (http://etl.snu.ac.kr/) .

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 can not 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 can not 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. Hence, before submitting your programming assignment, please be sure that your program makes correct answers in reasonable time for any input case.

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. His 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 can not find the book*".
- The operations of *BookSearch* should be fast to cover a lot of books in the library.

He also requested special types of search as follows:

- *Order search*: *BookSearch* should list all book names in lexicographical order.
- *First/Last search*: *BookSearch* needs to be able to find the first or last book in the library in lexicographical order.
- *Range search*: when two book names are given such as “bible” and “history”, it should search all books whose names are between “bible” and “history” in lexicographical order. If *BookSearch* contains “bible” and “history”, it also needs to find those books in the range.

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 a white space.
- *BookSearch* should be based on Binary Search Tree.
- Of course, you need to consider that the library contains a lot of books.

Make "*BookSearch*" class supporting those requirements. To do that, you need to fill in the "*BookSearch.java*" and the "*BST.java*" of "PA_02" java project. For convenience, we wrote incomplete codes of "Main" class to handle inputs, but you need to modify the "*Main.java*" to print proper outputs.

4. ADT of Data structure

1) add

Function
void add(String name, String location)
Description
- This function adds the book information into *BookSearch*. - The book information is in forms of a key-value pair: the key is “name” as the book name, and the value is “location” of the book.

2) remove

Function
void remove(String name)
Description
- This function removes the book information with “name” from *BookSearch.* - If the function tries to remove a book, and there is no book with the name, *BookSearch* should print the message “*BookSearch can not find the book*”.

3) get

Function
String get(String name)
Description
- Given the book name, this function should return the location of the book if *BookSearch* contains the information of the book. - If the function tries to search a book, and there is no book with the name, *BookSearch* should print the message “*BookSearch can not find the book*”.

4) size

Function
int size()
Description

- This function should return the number of books that *BookSearch* contains.

5) order

Function
void order()
Description

- This function retrieves the information of books in lexicographical order.
- The function should print all book names. Print each book name for each line.
- If *BookSearch* does not have any book, print the message "*BookSearch does not have any book*".

6) first

Function
String first()
Description

- This function should return the name of the first book in lexicographical order.
- If *BookSearch* does not have any book, print the message "*BookSearch does not have any book*".

7) last

Function
String last()
Description

- This function should return the name of the last book in lexicographical order.
- If *BookSearch* does not have any book, print the message "*BookSearch does not have any book*".

8) range

Function
int range(String from, String to)
Description
- (from) is the book name indicating the start of the range. - (to) is the book name indicating the end of the range. - This function retrieves all books whose name is between (from) and (to), including the book names (from) and (to) if they exist in *BookSearch*. - This function should print the number of books in the range.

5. Specification of I/O

1) add

Input form	Output form
`add (name) (location)`	`ADD: (name) (location)`

Description

- (name) is the name of a book to be added.
- (location) is the location of the book.
- (name) and (location) do not have any white space character within themselves for simplicity.

Example Input	Example Output
`add hunger_game A4-123`	`ADD: hunger_game A4-123`

2) remove

Input form	Output form
`remove (name)`	`REMOVE: (name)`

Description

- (name) is the name of a book to be added.
- If *BookSearch* tries to remove a book, and there is no book with the name, *BookSearch* should print the message "*BookSearch can not find the book*".

Example Input	Example Output
`remove hunger_game`	`REMOVE: hunger_game` `BookSearch can not find the book`

3) get

Input form	Output form
`get (name)`	`GET: (name) (location)`

Description

- (name) is the name of a book to be added in a word.

- (location) is the location of the book in a word.
- If *BookSearch* tries to search a book, and there is no book with the name, *BookSearch* should print the message "*BookSearch can not find the book*".

Example Input	Example Output
`get hunger_game`	`GET: hunger_game A4-123`
	`BookSearch can not find the book`

4) size

Input form	Output form
`size`	`CURRENT_SIZE: (# of books)`

Description

- (# of books) is the number of books in *BookSearch*.

Example Input	Example Output
`size`	`CURRENT_SIZE: 3`

5) order

Input form	Output form
`order`	`ORDER: (i-th book's name)`

Description

- For this input, the program should print all book names in lexicographical order. Print one book name per one line.
- (i-th book's name) is the name of i-th book in *the order search*.
- If *BookSearch* does not have any book, print the message "*BookSearch does not have any book*".
- Assume that for the below example, *BookSearch* has books as follows: { *history, romance, big_Little, becoming_nicole*}.

Example Input	Example Output

order	ORDER: becoming_nicole ORDER: big_little ORDER: history ORDER: romance
	BookSearch does not have any book

6) first

Input form	Output form
first	FIRST: (1st book's name)

Description

- For this input, the program should print the name of the first book.
- (1st book's name) is the name of the first book in lexicographical order.
- If *BookSearch* does not have any book, print the message "*BookSearch does not have any book*".
- Assume that for the below example, *BookSearch* has books as follows: { *history, romance, big_Little, becoming_nicole*}.

Example Input	Example Output
first	FIRST: becoming_nicole
	BookSearch does not have any book

7) last

Input form	Output form
last	LAST: (the last book's name)

Description

- For this input, the program should print the name of the last book.
- (the last book's name) is the name of the last book in lexicographical order.
- If *BookSearch* does not have any book, print the message "*BookSearch does not have any book*".
- Assume that for the below example, *BookSearch* has books as follows:

{*history, romance, big_Little, becoming_nicole*}.

Example Input	Example Output
`last`	`LAST: romance`
	`BookSearch does not have any book`

8) range

Input form	Output form
`range (from) (to)`	`RANGE: (# of books in range)`

Description

- For this input, the program should print the number of books in the range between (from) and (to).
- (from) is a string in lower case indicating the start of the range.
- (to) is a string in lower case indicating the end of the range.
- In lexicographical order, (from) must precede (to), or (from) can be equal to (to). For example, *range abc fbc*, and *range abc abc* are possible, but *range fbc abc* is not acceptable. (Hence, you do not need to consider the latter case of this input.)
- Assume that for the below example, *BookSearch* has books as follows: {*history, romance, big_Little, becoming_nicole, diary*}.

Example Input	Example Output
`Range becoming_nicole diary`	`RANGE: 3`

6. Sample Input and Output

The grading machine expects the sample output for given the sample input. Hence, for the sample input, your program should print the same lines in the sample output. If your program prints different lines from the sample output for the sample input, the grading machine will not give you the point of the input. See more details of the policy on the grading machine in Section 2.

1) Sample Input ('sample_input' file in 'PA_02.zip')

```
add history A4-123
add romance B5-331
add bible C2-112
remove bible
add big_little G5-123
order
add becoming_nicole D1-872
get big_little
first
last
add diary H9-100
range becoming_nicole diary
```

2) Sample output (‘sample_output’ file in ‘PA_02.zip’)

```
ADD: history A4-123
ADD: romance B5-331
ADD: bible C2-112
REMOVE: bible
ADD: big_little G5-123
ORDER: big_little
ORDER: history
ORDER: romance
ADD: becoming_nicole D1-872
GET: big_little G5-123
FIRST: becoming_nicole
LAST: romance
ADD: diary H9-100
RANGE: 3
```