Seoul National University

Data Structure

Fall 2017, Kang

Programming Assignment 2: Binary Tree (Chapter 5)

Due: Oct. 31, 14:00, submit at eTL

Reminders

- The points of this homework add up to 100.
- Like all homeworks, this has to be done individually.
- Lead TA: Jun-gi Jang (elnino9158@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_2017-12345.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_2017-12345.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 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. 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. Jun works as a teacher at a university. He organizes student's midterm grades. He needs to put student's grades when he completes grading. Also, grades can be discarded if the students drop the course. He wants to find the grade of the students when some students ask for their grades.

To manage midterm grades of students efficiently, he wrote down the information such as the student's name and the grade of a student on a paper.

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

- In *GardeSearch*, the information of the grade is stored in forms of a key-value pair. The key is the student's name, and the value is the grade of the student.
- For a student, Mr. Jun needs to be able to find the grade of the student using *GradeSearch* with the student name.
- Mr. Jun needs to be able to add or delete the information of students using *GradeSearch*.
- If Mr. Jun tries to search for or delete a student, and there is no student with the name, *GradeSearch* should print the message "*GradeSearch cannot find the student*".
- The operations of *GradeSearch* should be fast to cover a lot of students in the system.

He also requested special types of search as follows:

- *Order search*: *GradeSearch* should list all students' names and grades in lexicographical order.
- *First/Last search*: *GradeSearch* needs to be able to find the first or last

student in lexicographical order.

- *Range search*: when two student names are given such as “bob” and “harry”, it should search for all students whose names are between “bob” and “harry” in lexicographical order. If *GradeSearch* contains “bob” and “harry”, it also needs to find those students in the range.

Here are several assumptions for clarity.

- In the system, the student names are distinct. There is no duplicate of a student name.
- All student names are in lower case. Also, the student names do not have a white space.
- *GradeSearch* should be based on Binary Search Tree.
- Of course, you need to consider that the system contains a lot of students.

Make "*GradeSearch*" class supporting those requirements. To do that, you need to fill in the "*GradeSearch.java*" and the "*BinaryTree.java*" of "PA2" 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, int grade)
Description
- This function adds the student information into *GradeSearch*. - The student information is in forms of a key-value pair: the key is "student" as the student name, and the value is "grade" of the student.

2) remove

Function
int remove(String name)
Description
- This function removes the student information with "name" from *GradeSearch.* - If the function tries to remove a student, and there is no student with the name, *GradeSearch* should print the message "*GradeSearch cannot find the student*".

3) get

Function
int get(String name)
Description
- Given the student name, this function should return the grade of the student if *GradeSearch* contains the information of the student. - If the function tries to search for a student, and there is no student with the name, *GradeSearch* should print the message "*GradeSearch cannot find the student*".

4) size

Function
int size()

Description
- This function should return the number of students that *GradeSearch* contains.

5) order

Function
void order()

Description
- This function retrieves the information of students in lexicographical order. - The function should print all student names and their grades. Print each student name and grade for each line. - If *GradeSearch* does not have any student, print the message "*GradeSearch does not have any student*".

6) first

Function
String first()

Description
- This function should return the name and the grade of the first student in lexicographical order. - If *GradeSearch* does not have any student, print the message "*GradeSearch does not have any student*".

7) last

Function
String last()

Description
- This function should return the name and the grade of the last student in lexicographical order. - If *GradeSearch* does not have any student, print the message "*GradeSearch does not have any student*".

8) range

Function
int range(String from, String to)
Description
- (from) is the student name indicating the start of the range. - (to) is the student name indicating the end of the range. - This function retrieves all students whose name is between (from) and (to), including the student names (from) and (to) if they exist in *GradeSearch*. - If *GradeSearch* tries to search for a student, and there is no student with the (from) or (to) name, *GradeSearch* should print the message "*The range is wrong*". - This function should print the number of students in the range.

5. Specification of I/O

1) add

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

Description

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

Example Input	Example Output
`add bob 85`	`ADD: bob 85`

2) remove

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

Description

- (name) is the name of a student to be removed.
- If *GradeSearch* tries to remove a student, and there is no student with the name, *GradeSearch* should print the message "*GradeSearch cannot find the student*".

Example Input	Example Output
`remove bob`	`REMOVE: bob` *GradeSearch* `cannot find the student`

3) get

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

Description

- (name) is the name of a student to be added in a word.
- (grade) is the grade of the student in a word.
- If *GradeSearch* tries to search for a student, and there is no student with the name, *GradeSearch* should print the message "*GradeSearch cannot find the student*".

Example Input	**Example Output**
`get bob`	`GET: bob 85`
	`GradeSearch cannot find the student`

4) size

Input form	**Output form**
`size`	`CURRENT_SIZE: (# of students)`

Description

- (# of students) is the number of students in *GradeSearch*.

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

5) order

Input form	**Output form**
`order`	`ORDER: (i-th student's name) (i-th student's grade)`

Description

- For this input, the program should print all student names and grades in lexicographical order. Print one student name per one line.
- (i-th student's name) is the name of i-th student in *the order search*.
- If *GradeSearch* does not have any student, print the message "*GradeSearch does not have any student*".
- Assume that for the below example, *GradeSearch* has students as follows: {*bob-80, babi-90, henderson-70, romero-85*}.

Example Input	**Example Output**

order	ORDER: babi 90 ORDER: bob 80 ORDER: henderson 70 ORDER: romero 85
	GradeSearch does not have any student

6) first

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

Description

- For this input, the program should print the name and the grade of the first student.
- (1st student's name) is the name of the first student in lexicographical order.
- If *GradeSearch* does not have any student, print the message "*GradeSearch does not have any student*".
- Assume that for the below example, *GradeSearch* has students as follows: {*bob-80, babi-90, henderson-70, romero-85*}

Example Input	Example Output
first	FIRST: babi 90
	GradeSearch does not have any student

7) last

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

Description

- For this input, the program should print the name and the grade of the last student.
- (the last student's name) is the name of the last student in lexicographical

order.

- If *GradeSearch* does not have any student, print the message "*GradeSearch does not have any student*".
- Assume that for the below example, *GradeSearch* has students as follows: {*bob-80, babi-90, henderson-70, romero-85*}.

Example Input	**Example Output**
`last`	`LAST: romero 85`
	`GradeSearch does not have any student`

8) range

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

Description

- For this input, the program should print the number of students 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.)
- If *GradeSearch* tries to search for a student, and there is no student with the (from) or (to) name, *GradeSearch* should print the message "*The range is wrong*".
- Assume that for the below example, *GradeSearch* has students as follows: {*bob, babi, henderson, romero*}.

Example Input	**Example Output**
`range bob henderson`	`RANGE: 2`
	The range is wrong

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

```
add henderson 80
add romero 95
size
add babi 80
remove babi
add bob 90
order
add drogba 87
get romero
first
last
add dyer 93
range bob henderson
```

2) Sample output

```
ADD: henderson 80
ADD: romero 95
CURRENT_SIZE: 2
ADD: babi 80
REMOVE: babi
ADD: bob 90
ORDER: bob 90
ORDER: henderson 80
ORDER: romero 95
ADD: drogba 87
GET: romero 95
FIRST: bob 90
LAST: romero 95
ADD: dyer 93
RANGE: 4
```