Homework 2 - Implementing a *Skip list*

COP3503 assigned Feb 21, 2019
Michael McAlpin, Instructor due Mar 24, 2019

1 Objective

Build a *skip list* data structure to support the traversal, searching, addition, and deletion of integers from a *skip list*. This implementation will support building a *skip list* to support integers in the range of 0 to 10,000. The objective of this assignment requires the reading of an input file which contains commands and data to build a *skip list* containing integers using commands to insert, search, delete, and print a *skip list*.

2 Requirements

Read the input file formatted as follows. The input file will contain at least one command per line, either insert, delete, search, print, or quit. These are defined in detail below. And if appropriate, a second parameter may be required. The second parameter, if appropriate, will always be an integer for this assignment.

The commands are shown in the table below:

Command	Description	Parameter(s)
i	Insert	expects a space, followed by an integer
s	Search	expects a space, followed by an integer
d	Delete	expects a space, followed by an integer
p	Print	**does not** expect any additional data

Table 1: Input File Commands

2.1 Design Constraints

The input file(s) provided will have the following properties.

1. Each record in the input file will consist of a command, described above, appropriately followed by an integer.
2. There is not a ***maximum*** number of skip levels.
3. The test input integers will be within the range of 1 to 10,000.
4. There is no requirement for persistence. The data in the *skip list* does not need to be stored or archived on disk before the program exits.

2.2 Commands

2.2.1 Insert

The insert command uses the single character **i** as the command token. The command token will be followed by a single space, then an integer. This integer will then be inserted into the *skip list*. Note that a *skip list* requires that data be inserted into the *skip list* in ascending order.

1. Insertion of the lowest *rank* integer requires that this integer becomes the first element in the *skip list*. Note that **all** levels contain the *lowest rank integer*.

2. Insertion of an integer that is neither the *lowest* or *highest* rank integer requires a probabilistic mechanism to decide if this integer will also be *on the next higher level*. The method discussed in lecture is *flipping a fair coin*. The *flipping of a fair coin* can be emulated using the `Random` object in Java to generate a "random number" then taking that number **modulo 2** to generate a **1** or **0**. The optional seeding of the `Random` number generator will be the second command line parameter specified by an upper or lower case **R** . *In the event that the **R** parameter is **not** specified, seed the Random number generator with the integer **42**.* The promotion method used to generate the test cases used a **1** to represent *heads* and correspondingly a **0** to represent *tails*. Each flip of the coin that produces a *heads* causes that integer to be promoted and appropriately linked into the *skip list*. This process is terminated when a *tails* has been "flipped".

Inputs

`i xx` where **xx** is an integer between 1 and 10,000.

Outputs

`N/A`

2.2.2 Delete

The delete command uses the single character **d** as the command token. The command token will be followed by a single integer. In order to successfully delete an entry from the *skip list*, the intger must be found.

In the event that the *integer* cannot be found, the program will issue the error message `xx integer not found - delete not successful`, where `xx` is the specified integer. The program will recover gracefully to continue to accept commands.

Once the *integer* is found, it will be deleted from the *base* level, and any additional level(s) that integer had been promoted to upon insertion.

(This command's success can be verified by using the *print* command.)

Inputs

`d xx` where **xx** is an integer between 1 and 10,000.

Outputs

Success

`xx deleted` *:where xx is the integer being deleted*

Failure

`xx integer not found - delete not successful` *:where xx is the integer being deleted was not found*

2.2.3 Search

The search command uses the single character **s** as the command token. The command token will be followed by a single space, then the integer that is to be searched

The *search* command will take advantage of the *skip list* structure and implement the following algorithm. *A search for a target element begins at the head element in the top list, and proceeds horizontally until the current element is greater than or equal to the target. If the current element is equal to the target, it has been found. If the current element is greater than the target, or the search reaches the end of the linked list, the procedure is repeated after returning to the previous element and dropping down vertically to the next lower list.*[1]

Upon completion of the search for the target integer, the following messages shall be output.

Inputs

`s xx` where **xx** is an integer between 1 and 10,000.

Outputs

Success

`xx found` *:where xx is the integer being searched for*

Failure

`xx NOT FOUND` *:where xx is the integer being searched for and was not found*

2.2.4 Print

The print command uses the single character **p** as the command token. This command will invoke the *printAll* function described in detail below.
This command is critical for verification of all the commands specified above.

Inputs

`p`

Outputs

[1]Quoted from William Pugh's write up on Wikipedia.

```
ff210377;3.5;18.5
For the input file named in10.txt
With the RNG unseeded,
the current Skip List is shown below:
---infinity
 65;
 77;  77;
 90;
 120;
 450;  450;
 500;
 888;
 990;
 7000;
 7900;
+++infinity
---End of Skip List---
```

2.3 Classes and Functions

2.3.1 Classes

SkipList The *SkipList* class shall contain, at a minimum, the following methods.

insert The features and performance of the *insert* method is defined by the behavior described above.

promote The features and performance of the *promote* method is defined by the behavior described in the behavior of the insert commmand decribed above.

delete The features and performance of the *delete* method is defined by the behavior described above.

search The features and performance of the *search* method is defined by the behavior described above.

printAll The *printAll* method prints the contents of the whole *skip list* in the format specified below.

Additional methods and properties will be required to successfully implement the methods specified above.

2.3.2 Functions

complexityIndicator Prints to **STDERR** the following:

NID

A difficulty rating of difficult you found this assignment on a scale of 1.0 (easy-peasy) through 5.0 (knuckle busting degree of difficulty).

Duration, in hours, of the time you spent on this assignment.

Sample output:

```
ff210377@eustis:~/COP3503$ ff210377;3.5;18.5
```

3 Testing

Make sure to test your code on Eustis **even if it works perfectly on your machine**. If your code does not compile on Eustis you will receive a 0 for the assignment. There will be 10 input files and 9 output files provided for testing your code, they are respectively shown in Table 2 and in Table 3.

Filename	Description
input01.txt	Insert 7 integers with 2 searches and a delete the prints the *skiplist*.
in5.txt	Five integers inserted with no duplicates. Prints the *skiplist*.
in5del2.txt	Five integers added followed by two deletes. One will be a delete of a non-existent integer. Prints the *skiplist*.
in5del1srch1.txt	Five names added, one valid delete, followed by a valid search, then an invalid search. Prints the *skiplist*.
in10.txt	10 integers inserted with no duplicates. Prints the *skiplist*.
in100.txt	100 integers inserted. Prints the *skiplist*
in100m5000.txt	100 random integers (all modulo 5,000) inserted with random deletes.
in10k-m5000.txt	10,000 integers (all modulo 5,000) inserted with random deletes.
in1m-m5000.txt	1,000,000 integers (all modulo 5,000) inserted with random deletes.
mega5.txt	5,000,000 random integers with random deletes. (***For entertainment value, as it takes a long time to complete***.)

Table 2: Input files

The expected output for these test cases will also be provided as defined in Table 3. To compare your output to the expected output you will first need to redirect *STDOUT* to a text file. Run your code with the following command (substitute the actual names of the input and output file appropriately):

```
java Hw02 inputFile > output.txt
```

The run the following command (substitute the actual name of the expected output file):

```
diff output.txt expectedOutputFile
```

Make sure that the `Random Number Generator` is NOT seeded. That is, do not use the **r** option when testing.

If there are any differences the relevant lines will be displayed (note that even a single extra space will cause a difference to be detected). If nothing is displayed, then congratulations - the outputs match! For each of the five (5) test cases, your code will need to output to *STDOUT* text that is identical to the corresponding *expectedOutputFile*. If your code crashes for a particular test case, you will not get credit for that case.

4 Submission - via WebCourses

The Java source file(s). Make sure that the *main* program is in Hw02.java.

Use reasonable and customary naming conventions for any classes you may create for this assignment.

5 Sample output

```
ff210377@eustis:~/cop3503/hw2 $ java Hw02 in10.txt
ff210377;3.5;18.5
For the input file named in10.txt
With the RNG unseeded,
the current Skip List is shown below:
---infinity
 65;
 77;  77;
 90;
 120;
 450;  450;
 500;
 888;
 990;
 7000;
 7900;
+++infinity
---End of Skip List---
ff210377@eustis:~/cop3503/hw2 $ java Hw02 in10.txt >in10St.txt
ff210377;3.5;18.5
ff210377@eustis:~/cop3503/hw2 $ diff in10St.txt in10Valid.txt
ff210377@eustis:~/cop3503/hw2 $
```

*Note: This is based on an actual **skip list** output using the inputs specified in the commands shown above.*

Note The **ff210377;3.5;18.5** output shown above is the output from the *complexityIndicator* function to **STDERR**.

Command	Validly formatted output files
java Hw02 in5.txt > in5St.txt	in5Valid.txt
java Hw02 in5del1srch1.txt >in5del1srch1St.txt	in5del1srch1Valid.txt
java Hw02 in5del2.txt >in5del2St.txt	in5del2Valid.txt
java Hw02 in10.txt > in10St.txt	in10Valid.txt
java Hw02 in100.txt > in100St.txt	in100Valid.txt
java Hw02 in100m5000.txt > in100m5000St.txt	in100m5000Valid.txt
java Hw02 in10k-m5000.txt > in10k-m5000St.txt	in10k-m5000Valid.txt
java Hw02 in1m-m1000.txt > in1m-m1000St.txt	in1m-m1000Valid.txt

Table 3: Commands with input files and corresponding output files.

6 Grading

Grading will be based on the following rubric:

Percentage	Description
-100	Cannot compile on *Eustis*.
- 80	Cannot read input files.
- 25	Cannot insert an integer into *skip list*.
- 25	Does not build a valid entry for the lowest/highest integer in the *skip list*.
- 25	Cannot promote integer up one level using a *fair coin* flip.
- 25	Cannot search for an integer in the *skip list* correctly. This includes a search for a non-existent integer.
- 25	Cannot print the contents of the *skip list* correctly.
- 25	Cannot delete an matching entry in the *skip list* correctly. This includes the error case of correctly handling an attempted delete of a non-exstent integer.
- 25	Does not support the *random* option of either an upper or lower case **R** to seed the random number generator and produce different *skiplists* based on a different seed value.
- 10	Output does not match *expectedOutput.txt* exactly.

Table 4: Grading Rubric