COMP202 SPRING 2016

Programming Project#3

Due Date: Wednesday **April 6 at 23:59**

Submission Location: F Drive

In this project, you are investigating a Skip List data structure. You are going to build a Skip List with a set of numbers from the input, visualize it, do a search for an item and save/retrieve it from/to a file. The important part is to visualize the Skip List data structure that you have built based on the input data on the console.

Input:

Your project should read the input from the input.txt file which is located in the same directory of your project. The elements are placed in a line and are separated from each other by a single space. In this project, for the sake of simplicity, all the input elements are assumed to have only two digits. In the other word, all the input numbers are in the range of [10,99]. There exists only a single number in the second line. That number represents the capacity of the Skip List as the maximum number of possible elements. A sample input structure for a Skip List with 7 elements and maximum capacity of 128 elements is as the following:

```
10 73 22 19 15 45 37

128
```

Building the Skip List:

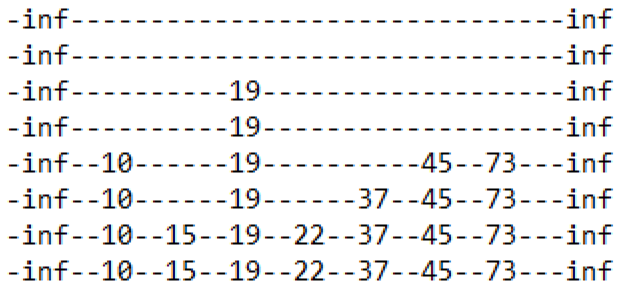
You are supposed to follow the original insertion algorithms of the Skip List (from the textbook) and insert the input numbers one by one in the Skip List in order to build it. However, there is a minor difference between the original insert algorithm and the one we expect from you. Instead of determining the height of an element's tower via tossing a coin repeatedly, you should use the following height length generator:

For a Skip List with the maximum capacity of N elements, define $m = ceiling(log_2 N)$. *Now for each element e, define the height of e as h(e) = e mod m.*

For the input example, we have N = 128 and m = 7. Therefore, the height of an element's tower in this Skip List is defined as h(e) = e mod 7. For instance h(10) = 3. Which means that element number 10 has a tower from level zero up to level 3.

Visualizing your output:

After you built your Skip List based on your input data, you need to visualize your Skip List in the console consistent with the example that is shown below. Note that for a certain m, we have m+1 levels in the Skip List numbered from 0 to m.



```
-inf-------------------------------inf
-inf-------------------------------inf
-inf----------19-------------------inf
-inf----------19-------------------inf
-inf--10------19----------45--73---inf
-inf--10------19------37--45--73---inf
-inf--10--15--19--22--37--45--73---inf
-inf--10--15--19--22--37--45--73---inf
```

Figure 1- Visualization of the sample Skip List based on the input data

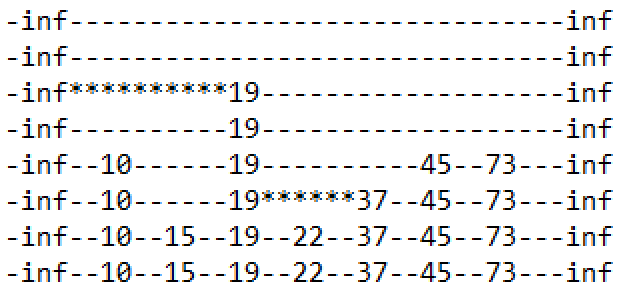
While you are developing your visualization function, you are expected to observe the following rules:

1- Insertion of –inf and inf at the first and last column with the height of m.
2- Having only –inf and inf in the topmost level.
3- For a certain element with a defined h(e), it should have elements from level 0 to level h(e).
4- All the elements should be sorted in the ascending order in level zero.
5- Between each two consecutive elements in the visualization 2 dashes (--) should be placed.
6- All the elements of the same tower should be adjusted horizontally.

Search and insert operations:

After your Skip List has been built based on the input file as well as visualized in the console, your program should listen to the user's keyboard input. User can perform search or insert via keyboard as <operation>*<value> pairs command.

For search, your program is **JUST** expected to visualize the search path. An example of search command is: search*39 and the expected output is as the following:

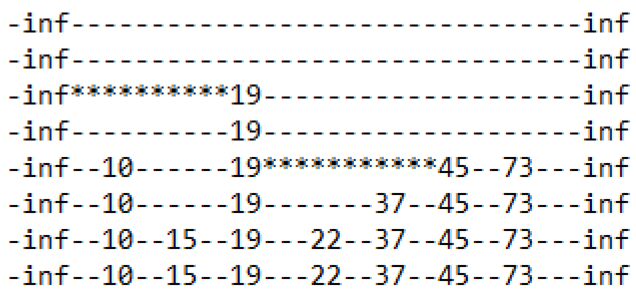


```
-inf-------------------------------inf
-inf-------------------------------inf
-inf**********19-------------------inf
-inf----------19-------------------inf
-inf--10------19----------45--73---inf
-inf--10------19******37--45--73---inf
-inf--10--15--19--22--37--45--73---inf
-inf--10--15--19--22--37--45--73---inf
```

Figure 2- A search example for 39

Please notice that the search algorithm in a Skip List returns the greatest element that is less than or equal to the search target. As it is shown in Figure 2, your program is expected to show the search path by * while obeying the visualization rules of Skip List.

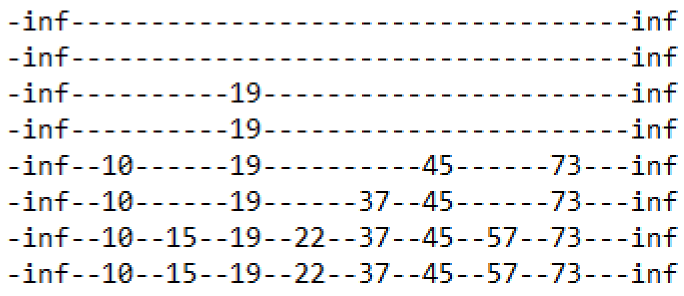
For insertion, you are expected to first perform a search for the insertion value and visualize the search. After the search was performed, you need to insert the value into the Skip List and visualize the final Skip List after insertion. An example of insertion command is insert*57. Figure 3 shows the first part of the insertion operation, which is the search for the inserted value (search for 57). As it is shown in this figure, the greatest element in the Skip List which is less than or equal 57, is 45. The path from the root element to 45 is labeled by * as the first part of expected output.



```
-inf--------------------------------inf
-inf--------------------------------inf
-inf**********19--------------------inf
-inf----------19--------------------inf
-inf--10------19***********45--73---inf
-inf--10------19-------37--45--73---inf
-inf--10--15--19---22--37--45--73---inf
-inf--10--15--19---22--37--45--73---inf
```

Figure 3- Insertion of 57 into the Skip List: First phase is to perform a search for it and visualize the search path as output

Figure 4, shows the second part of the insertion operation which putting the 57 right after 45. Since 57 mod 7 = 1, this new element has a height of 1 and is located in the zero and first levels.



```
-inf-----------------------------------inf
-inf-----------------------------------inf
-inf----------19-----------------------inf
-inf----------19-----------------------inf
-inf--10------19----------45------73---inf
-inf--10------19------37--45------73---inf
-inf--10--15--19--22--37--45--57--73---inf
-inf--10--15--19--22--37--45--57--73---inf
```

Figure 4-Insertion of 57 into the Skip List: the Second phase is to put the element into the Skip List and wire the links

In summary, as the result of a search algorithm, you are expected to output the Figure 2, and for the case of an insertion, you are expected to output the figure number 3 and 4, respectively. When one operation finishes completely, you are expected to listen back to the user's keyboard for another possible operation.

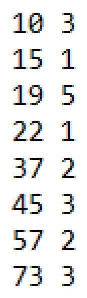
I/O operation,

Your program is expected to be able to store and retrieve a Skip List data structure in/from a text file. A save command is in the form of: save <file name>. After the user enters this command, your program should save the Skip List nodes in the form of <element value> <height> in a text file which its name is determined by <file name> field. After save was done, your program is expected to show a "Skip List has been saved successfully" message.

As an example, user may save the Skip List that is represented by Figure 2 in a list1.txt file by writing the following command:

save list1.txt

After that, in your root directory, you should have a list1.txt. Content of that file is expected to be exactly like this:



```
10 3
15 1
19 5
22 1
37 2
45 3
57 2
73 3
```

Figure 5-Content of list1.txt file

A retrieve command is in the form of: load <file name>. After execution of this command, your program should read the input file and load the data into nodes of a Skip List without performing any insertion operation. Finally, your program should visualize the loaded Skip List.

Note: After loading a Skip List by load operation, the current Skip List will be substitute with the loaded one.

Important Implementation Notes:

1. Please read this project description carefully before doing any implementations.
2. Please note that you need to build a Skip List from the input.txt file at the beginning of your program. However, for the case of load, no build is needed as the elements are sorted and their height is already given.
3. You should not use any random generator rather than the one we asked you to build as the height generator.
4. You should implement the Skip List by yourself and utilization of any data structure related library except string processing and file related libraries are **strictly forbidden, and consequence a grade of zero for this project.**
5. In general, after reading a Skip List from the input.txt file and building and visualizing that, your program should repeatedly listen to I/O operations, search or insert.
6. **After loading a Skip List by load operation, the current Skip List will be substitute with the loaded one.**
7. You should implement your whole project by only 5 classes: main, node, skiplist, operations, and visualization. For each class, all of the data members should be defined as private and have their own getter and setter. **Please use the exact naming and rules. Otherwise, your project will not be graded**. The descriptions of this classes are as the followings:
    a. node: represents a single node of Skip List, includes its data members like value, right neighbor, left neighbor, height, etc.
    b. skiplist: represents a Skip List as a set of nodes accompanied by the insertion and search.
    c. operations: handles the I/O operations.
    d. visualization: handles the visualization.
    e. main: only should contain your main function.
8. You may comment where it is necessary to explain your implementation. Comments are not compulsory, but a good commenting skill will take into account while grading. Since there is not demo

for your project, comments are the only way you can justify your implementations, so please take it seriously!

Important Submission Rules:

Please **do not zip your files** and follow the submission rules. The projects uploaded different than given format **will not be graded.**

Upload it through Novel Login to the F drive/COURSES/UGRADS/COMP202/PROJECT/YOUR_NAME

Please consider the following format for your projects submission:

Project3 “This should be only the name for your folder”

Do not upload two versions of your project under the same name, instead keep a counter and add it to the end of your file:

first version : Project3
second version: Project3_V2
ith version: Project3_Vi

We only check the latest version of your submissions.
Except the emergency situation, please try to upload JUST one and the final version