

Lab 3

March 11, 2003

Due Date: Tuesday March 25

In this lab you will implement a basic system that stores historical events where each event is specified by a date (just the year) and a description. You will use a skiplist with the date as a key to support a query that lists in sorted order (by date) all events between two specified dates. In order to receive an “A” for the lab, you will also use your hash table from Lab 2 (or the hash table built into Java) to support an operation in which the user gives a word and receives a sorted list of events that contain the given word. Remember that you receive a 10% bonus for whatever portions you hand in by March 18 at the start of class.

There are many methods others than those requested here that you would want to add in a commercial product. For the purposes of this lab I’ve tried to remove most of the extras with a focus on those that are fundamental to understanding a skiplist. Note that there is flexibility provided to you as far as how you want to implement these. You are required to submit a discussion about your design choices.

Part 1: Core Skiplist Methods (50 pts)

Implement the `put`, `toString`, and `containsKey` methods described below. For this initial portion you can define the head and tail towers of your skiplist to have a height of 100 (limiting the maximum height of any item to 100). You should have an instance variable that keeps track of the highest level in use and start each search (including those used within `put` and `containsKey`) at this level.

`void put(int date, String description)`: Inserts an item into the skiplist with the key of `date` and associated data of `description`. You should expect to have multiple historical events inserted with the same date and you are expected to store them in the skiplist. One of the decisions you need to make is whether or not you want to allow duplicate keys or instead keep a list of events for each key.

`toString` (or in C++ overload the `<<` operator) which takes no parameters and returns a string (or `ostream` in C++). For this lab, the `toString` method will be set up to demonstrate that you have implemented the skiplist correctly. This will help you to see if you have made a fundamental error in your skiplist implementation so that you can correct the problem before the lab is submitted. The expected return value should be something like the below. You MUST output level `i` by traversing the `next[i]` pointers.

```
Level 4: 1663
Level 3: 1450 1663 1958
Level 2: -3500 1450 1663 1876 1877 1928 1958
Level 1: -3500 105 1450 1454 1663 1876 1877 1928 1931 1949 1958
Level 0: -3500 -1500 -320 -170 -134 105 1450 1454 1658 1663 1831 1835
1876 1877 1925 1928 1931 1942 1949 1958
```

When not in debugging mode, you would really just want to output level 0 but for this lab you can assume that printing of the skiplist is only used for debugging.

`boolean containsKey(int date)`: Returns true if any historical event with the given date is contained in the skiplist. If no event has the given date then false is returned. For the purpose of this lab, every time `containsKey` is called should print which comparisons are

made at each level. For example, when given `containsKey(-134)` in the skiplist shown above, `containsKey` should output what is shown below with `false` returned. The last line shown is output by the driver.

```
At level 4 compared -134 to: 1663
At level 3 compared -134 to: 1450
At level 2 compared -134 to: -3500   1450
At level 1 compared -134 to: 105
At level 0 compared -134 to: -1500   -320   -170   -134
The date -134 is in use.
```

You are expected to use this output to confirm the skiplist is functioning properly. There are many wrong skiplist implementations that I've seen students use (e.g. performing the entire search at level 0) that would lead to the correct answer but would take away the efficiency that skiplist provide. Again, in reality, you would only do this printing in a debug mode but for the purpose of this assignment, you should can assume `containsKey` is only called in debugging mode.

Note that the `put` method should follow the same path as `containsKey` in finding where to perform the insertion. The key differences is that you splice the new tower in at each level at or below the height selected for the new tower as you go down. There will be a significant deduction if `put` is done in such a way that would not have expected $O(\log n)$ time. Here's a concrete example of a common error I've seen in the past. Suppose that you are inserting a new item and its size (as selected by the random process described in class) is 1 (so it only will be inserted in L_0). I've seen many students then search for the item's position using only list L_0 and then insert it in the appropriate place. While the skiplist will have the proper structure, this error causes `insert` to have time complexity of $O(n)$ which makes this very inefficient.

On the web page under Lab 3, you will find some useful code fragments for reading in the events and doing the parsing. Also, a method (to be incorporated as part of your `Skiplist` class) to simulate a fair coin flip is given there.

Part 2: Implement the Range Search Method (15 pts)

Instead of a standard search method, you are to implement the range search described below. Think about the most efficient way to implement this method using your skiplist. Be sure to describe how you did this (and analyze its asymptotic time complexity) in your write-up.

`rangeSearch(int startDate, int endDate)`: This method needs to either return a list of historical events (or a string to output) containing all historical events with a date `d` satisfying `startDate <= d <= endDate`. Furthermore, this list should be sorted by the date (with ties broken arbitrarily for events with the same date). Here is sample output from my driver.

```
Options: Word Search(1), Range Search(2), Remove(3), Exit(4): 2
Start Date: -5000
End Date: -2000
-4000: Copper metallurgy is invented and copper is used for ornamentation
-3500: River boats are invented
-3500: Wheeled carts are invented
-3500: The Sumerians develop cuneiform writing and the Egyptians develop
hieroglyphic writing
-3000: Bronze is used for weapons and armor
-3000: Candles are invented
-2136: Chinese astronomers record a solar eclipse
-2000: Horses are tamed and used for transport
```

Part 3: Implement the Remove Method (10 pts)

Here you are to add the following remove method. In your write-up summarize how you implemented this.

`remove(int date)`: Removes all events with the given date from the skiplist. Return either a list of historical events removed or if you prefer you can return a String that includes the descriptions of all the events removed.

Part 4: Doubling the Head and Tail Towers (5 pts)

Do not make any assumptions about the maximum height of an item. You must begin with the head and tail towers each having size 2 (so there are only two levels, level 0 and level 1). Then if during insert you randomly select the size of a node to be larger than your current size of the head and tail, then you should double the size of your head and tail. Don't be surprised to find that you don't need to do this very often. Notice if you increased the size of head and tail by just one element then you would be okay (on average) until the number of items doubled in value. Thus by doubling the size of head and tail you don't expect to do this again until the number of elements is n^2 where n was the number of elements when you doubled the size of head and tail.

To receive full credit here, you must also keep track of the largest level which was ever selected so that all searches can begin at that level.

Part 5: Use a HashTable to do the Word Search (20 pts)

Use a hashtable to efficiently support `findOccurrences(String word)` that returns a list of all events containing `word`. (See Homework 2, Problem 4a). To make this method not sensitive to case you can use `word.toLowerCase()` to convert the word to lower case for the key (i.e. the word) for both your hash table `put` method and for the `findOccurrences` method. Here is sample output from my driver.

```
Options: Word Search(1), Range Search(2), Remove(3), Exit(4): 1
Word: last
1637: Pierre de Fermat claims to have proven Fermat's Last Theorem in his copy
of Diophantus' Arithmetica
1825: Peter Dirichlet and Adrien Legendre prove Fermat's Last Theorem for n=5
1832: Peter Dirichlet proves Fermat's Last Theorem for n=14
1966: Rainer Sachs and Arthur Wolfe theoretically predict microwave background
fluctuation amplitudes created by gravitational potential variations between
observers and the last scattering surface
1983: Gerd Faltings proves the Mordell Conjecture and thereby shows that there are
only finitely many whole number solutions for each exponent of Fermat's Last
Theorem
1993: Andrew Wiles proves part of the Taniyama-Shimura Conjecture and thereby proves
Fermat's Last Theorem
```

There are about 4000 words (the exact number varies depending on if you convert all words to lower case and whether or not you include punctuation as part of the word, etc). Feel free to use this known value in initializing your hashtable size. If the same word is in an event more than once then it is okay for the purpose of this assignment to include it for each occurrence of the word but it would be nicer to only include each event once per word it contains. Since you must output the events with a given word in sorted order, you probably want to keep your list of events associated with a date sorted. You can assume that there are few enough events associated with each data

that just a sorted list can be used. However, if you'd like, you could instead use your skiplist class here. Clearly describe what implementation choices you made in your write-up.

Note: If you implement both the remove and word search methods then you need to update the hashtable used for `findOccurrences` when an event is removed. 5 of the 20 points are associated with modifying your `remove` method to update the hashtable.

What to Submit

For this lab you are expected to submit the following. Please include these items in the given order so that the TAs can find everything easily.

- A completed and signed cover sheet with your name written legibly on the top.
- A write-up along the lines for Problem 4 of Homework 2 describing how you designed your lab. Include a brief discussion for each operation you support. Be sure to analyze the asymptotic time complexity of the `rangeSearch` method where n is the number of events stored and m is the number of dates that fall into the given range. Also address how you handled multiple events with the same key for your skiplist and how you handled multiple events having the same word in your hashtable (if you did Part 5).

If there were any problems with your implementation (e.g. wrong output was obtained, a run-time error occurred) then clearly indicate that in your write-up and give as much information as you can as to what you think is causing the problem.

- Your code written for Lab 3. You do not need to include code from Lab 2. (If your lab 2 did not work properly, use Java's built in `HashSet` or `HashMap` classes.)
- Using the `20events` data¹, insert all events into your skiplist (using `put`). Then print the skiplist. Next call `containsKey` three times, with the dates: -134, 1942, 1776.

If you implemented `remove` then call it three times with the dates: -3500, 1958, and with some date that is at the highest level in use. Then print the skiplist again.

I would recommend you create a very simple driver that calls the 20 inserts and then allows the user to either call `containsKey`, `remove` (if you implemented it), or exit. If you are having trouble with reading in the events or with the I/O please see us for help since you are not expected to need to spend very long with these aspects. Also don't forget to look at what is provided on the course web page under labs.

- Using the `allEvents` data set, insert all events into the skiplist using `put`. Next call range searches with the following 5 ranges: -5000 to -2000, 1776 to 1776, 1945 to 1945, 0 to 100, and 1770 to 1779.

If you implement the word search portion (Part 5), then show the results of searching individually for each of the words, "last" (if you do not convert this to lower case then use "Last"), "virus", "root", and "pi".

Finally, if you implemented `remove` then remove all events with the date 1771 and output the list of descriptions that is returned. Then repeat the range search from 1770-1779, and perform a word search on "dioxide" and "nebulae" to confirm that it works properly.

¹Both data sets were obtained using material I found on the web that has a copyright by Niel Brandt, 1994. Reproduction and distribution for personal profit is not permitted. To bring the `allEvents` data set to 1000 events, I added some events mostly from the Washington University Sesquicentennial web page.