Lab 8: Don't Skip This Lab

Pre-Lab Notes

Lab submissions will **not be accepted late** – they must be submitted by **midnight the day after they're assigned** (to account for time zone issues). Some latitude might be allowed (I won't disqualify a submission that's mere minutes late!) but that's purely discretionary.

Labs are treated differently from assignments – you're encouraged to work with fellow students, ask questions of the TA during your lab section, and generally collaborate to complete your work. You should still submit your own work, **including citing with whom you collaborated in your submission**, but using answers developed by others will not be treated as plagiarism.

Nevertheless, you are still encouraged to make a good faith effort to complete your labwork, in order to exercise your understanding of the topics it covers. Lab submissions are marked purely 1 or 0 by whether the TA believes you at least made an honest attempt, even if you didn't succeed, so the value is in what the attempt teaches you rather than simply having the right answer.

1. Ordered Maps and Skip Lists

The middle chapters of the course so far have been dominated by introducing alternatives to some of the underlying assumptions of the earlier data structures. Trees gave us non-linear data structures, Priority Queues introduced us to non-positional, key-based structured, and Heaps combined the two. Maps presented whole new possibilities opened up by these changes, and Hash Tables introduced us to some of the complicated math we can use to make the most of those opportunities.

However, lately we've been coming around full-circle to some of the concepts we were moving away from. Adaptable Priority Queues reintroduced position through location-aware entries, while Hash Tables essentially turned each unique key into a separate address in an array. This trend of going the long way to reproduce structures and mechanisms we started out with continues with Ordered Maps and Skip Lists, where we enforce a full ordering of entries according to their keys and start exploiting their relative positions to one another all over again.

Skip Lists, like Heaps, are mostly useful as a way to implement other data structures. They can be a little confusing, though – not just the theory of why they work or what they're accomplishing, but even how to build one to begin with! Fortunately, we're going to get you started on building your own Skip List today, so that'll get you started toward using them tomorrow!

2. Outline

I glossed over a lot of implementation work during the lecture on Skip Lists that you'd need to do in order to actually build a randomly-generated two-dimensional quasi-grid of linked list entries. Building a fully-featured Skip List class, complete with Skip Search, insertions, and removals, would be a tall order for just one lab. Instead, we're just going to focus on a single piece of the puzzle – the constructor.

You're going to build a constructor for a Skip List that takes in an array of integers and then builds a Skip List based on it according to the process I laid down during the lecture. To keep it simple, we'll say these integers come to you pre-sorted and will be randomly rolled between 1 and 100 (so header and trailer nodes of 0 and 101 at the start and end of each level can contain them).

Here's a few pointers to help get you started. First off, it's a good idea to build a node class to store your integer element along with above, below, next, and previous links – even if you could work around needing them in the constructor, you'd want these links in the future for Skip Search, so you might as well include them now.

You may be able to use a LinkedList or ArrayList to represent each level, as well as the list of all levels. I'd advise against a primitive array, however, especially if you want your Skip List to be able to grow or shrink in the future. You should also avoid solutions that leave big gaps of empty memory or include many null nodes, as the waste isn't worth spacing out each level to the same length.

It might be tempting to try some recursion here, but I actually wouldn't advise it – building iteratively, one layer at a time, will be a little easier and more straightforward. You still might want to make your work during each loop easier by keeping a reference to a previousNode and/or previousLayer that you can use for linking up the next/previous and above/below links, though.

Remember, ultimately the goal is to create randomly-chosen subsets of the original list connected through the duplicate entries. Whatever shortcuts you take, make sure those ideas remain intact, and don't forget to test your coin-flipper!

2.1 Show Your Work

To test your constructor, build a simple printing function that prints the lists to the terminal, one line per level. While it might be nice to properly space out the towers the way they appeared in lecture, it's also fine to just go ahead and print the contents of each list as its own row just to let you check that your Skip List was built correctly.

Write a main method that includes an array with ten numbers you give to your class constructor and then call your print function. Go ahead and try running that main method a few times and watch the results fluctuate. If you're really feeling ambitious, you could even try generating some stats by randomly generating a bunch of ten-int arrays and collecting averages and the like, but that's beyond the scope of today's simple exercise. Still, a good first step toward full Skip List-ing!

3. Deliverables

All lab submissions will be done through CourSys at https://courses.cs.sfu.ca/. For this lab, that should include:

1. A .zip archive of your code, organized into a single Java project. To ensure compatibility, it's recommended to use IntelliJ while developing your code, then using the export function under File -> Export -> Project To .zip File. **Don't forget to include a test file**, a Main.java with a main method that prints the results of a few runs of your functions to the terminal would be best.

Be sure you tidy up your code before submitting! Check the Java code style guide posted on the course website, and do your best to provide helpful comments.