CE204 Lab 4: Priority queues

David Richerby

Week 20, 2020–21

The lab exercises are not assessed and you are not required to complete all of them, though I recommend that you attempt them all. Feel free to work with others and talk about your answers.

Solutions will be released on Moodle, on the Friday after the labs.[1]

1 Sorting with a priority queue

Write a method `static void sort(int[] ints, boolean ascending)` that sorts an array of integers into ascending or descending order, using a priority queue.

What is the running time of your code?

2 Extra operations

Implement the following operations in a priority queue.

a) `void changePriority(int oldP, int newP)` finds a node with priority `oldP` and changes it to `newP`. If this violates the heap property, you will need to bubble the node up or down to fix it. To find the node, there's nothing significantly better than just searching through the array (why not?). There might be more than one node with the same value; you can change the first one you find. You should extract the bubbling up/down code from `insert()` and `next()` into separate private methods, to avoid duplication. What is the running time of your code?

b) `void delete (int p)` deletes a node with priority `p` from the heap. You'll need to replace it with something and fix any violations of the heap property. What is the running time of your code?

c) `int insertThenNext (int p)`. This should have the same effect as first calling `insert(p)` and then calling `next()`. However, the implementation should be more efficient. After you've removed the value at the root,

[1] I promise. They're uploaded already!

you can replace it with `p` instead of with a leaf. Does `insertThenNext()` have better running time than `insert()` followed by `next()`? Is it faster?

3 Heapsort

Heapsort is a more memory-efficient way to sort with a priority queue than the one you produced for the first exercise. For exercise 1, you will have copied all the elements of the input array into a separate array that stores the binary min-heap. Heapsort avoids the need to create a separate array. In this exercise, you'll write a version of heapsort that sorts arrays into descending order. This is more conceptually challenging than the other exercises.

We will build up the heap inside the original array. When you create an ordinary binary min-heap, the constructor creates an array to store the heap in. Initially, the array will be filled with zeros, but you don't care what's in there. More specifically, the algorithms for `insert()` and `next()` won't touch any array cells that don't store data that's in the heap. For example, if you store a heap in an array of size 10 and add and remove numbers from the heap in such a way that the size is always at most five, cells 5–9 of the array will never be touched.

So, suppose you're given an array of ten `int`s. You can say that the first cell of the array represents a binary min-heap containing just one number, and the rest is whatever. Now consider the first two cells together. You can imagine that the value in the second cell is there because you just started inserting it into the heap: you've placed it in the position where the next leaf would go, and you were just about to compare it with its parent and bubble it up, if necessary. So do that. Now, the first two cells are a valid heap and you can imagine that you just started to insert the third item by placing it where the next leaf would go. You can keep doing this for all of the cells in turn. By the time you've finished, the whole ten-element array is a binary min-heap, containing all the values that were originally stored.

Now, we take things out of the heap. Call `next()` to get the smallest element. Now, cells 0–8 of the array are a heap and cell 9 is free. Put the smallest element in it. Call `next()` again: you get the second-smallest element of the original array, and cell 8 is now free. You can keep calling `next()` to fill up the array in ascending order from the back – which is descending order from the front. By the time you've finished, the array is sorted in descending order.

a) Implement heapsort.

b) How would you modify the algorithm so that it sorts in ascending order without losing efficiency?