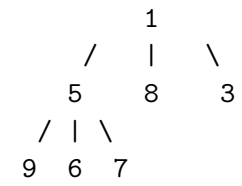
Compuer Science 62
Assignment 8

Due 11:59pm on Tuesday, April 6, 2010

In this assignment we will be playing around with heaps other than binary heaps and the effect that this has on memory usage and run-time.

n-ary heaps

So far, all of the heaps we have looked at have been binary heaps where each node has two children. As mentioned in class, we can also think about n-ary trees where rather than having two children, each node has n children. An n-ary heap, is a heap that is based on an n-ary tree. For example:



is a 3-ary heap. The definition of the heap remains the same where a parents value must be smaller than or equal to that of it's children and the tree must be complete. The only difference is that rather than having two children, each node has n children.

As with a binary heap, you can implement an n-ary heap using an `ArrayList` to store the data. The heap above would be stored as:

```
[1, 5, 8, 3, 9, 6, 7]
```

You are to implement a new class that represents a heap as an `ArrayList` but has an addtional parameter to the constructors n that allows you to build

an n-ary heap, rather than just a binary heap. The class should be defined as:

```
ArrayListNPriorityQueue<E extends Comparable<E>> implements PriorityQueue<E>
```

You need to implement all of the required methods for the `PriorityQueue` interface. In addition, you should provide two constructors, one that builds an empty n-ary heap:

```
public ArrayListNPriorityQueue(int numChildren)
```

and one constructor that builds an n-ary heap from data:

```
public ArrayListNPriorityQueue(ArrayList<E> data, int numChildren)
```

This constructor must use the `heapify` approach for building the heap rather than simply adding the data items.

Memory and Time

Once you have this working, I'd like you to investigate how the size of n in your n-ary heap affects the performance of the heap. To do this, I'm going to have you experiment with the data and then give a short write-up explaining your results.

First, experiment how the size of n changes the memory usage and runtime of the heap. In particular, vary n from 2 to 20. For each n, see how much memory is used to store the heap (using our `Memory` class used in the lab) then see what how long it takes to remove all of the elements from the heap using calls to `extractMin`. For each n, average the results over 10-20 runs to get better data. A heap of size 10,000 elements is big enough to get good data, but not so large that it is too computationally expensive.

Work towards generating a table that looks something like:

```
degree |           time |         memory |
     2 | <some_number> | <some_number> |
     3 | <some_number> | <some_number> |
     4 | <some_number> | <some_number> |
       ...
```

Once you have this data, think about what is happening and why. In addition to submitting the code above for the `ArrayListNPriorityQueue`

class, you also need to submit a short write-up. Your write-up should include:

- Your table showing your data.
- One paragraph explaining what you see in your data and why you see that behavior. You should talk about both memory usage and run-time.

Getting started

I've included copies of the `Memory` class, the `StopWatch` class and the `PriorityQueue` interface at:

`/common/cs/cs062/assignments/assignment8`

Make sure to use this `PriortyQueue` interface and not the one in bailey's library (you shouldn't need bailey's library at all for this assignment).

Submission

As usual, make a jar of your code (in this case, it will just have the one file: `ArrayListNPriorityQueue`). I do not need to see your code for generating your data. Put your jar file and your writeup in the folder that you will be submitting with your name on it.

Write your code carefully and clearly. Follow the specifications precisely, because your classes may be tested in ways that you do not suspect. As always, we will look at your Javadoc documentation. Submit the final versions of these files, with the names specified, in the usual way.

Grading

You will be graded based on the following criteria:

criterion	points
n-ary heap	10
data and write-up	6
appropriate comments (including JavaDoc)	2
style and formatting	1
submitted correctly	1

A few hints

Rather than have a method `left` and `right` (which don't make sense anymore for an n-ary tree) it may be useful to have a new method call `child` that takes the current node and also a child number and returns the index of the jth child of the current node.

Also, look (and utilized) the `ArrayListPriorityQueue` code we discussed in class. While not exactly the same, your n-ary heap will look somewhat similar.