Homework 11: Trees

Due: 10:00 PM, Nov 28, 2017

Contents

Objectives

By the end of this homework you will be able to:

1. Show that logarithmic functions are in the same big-theta class
2. Find paths down trees

How to Hand In

For this (and all) homework assignments, you should hand in answers for all the non-practice questions. **For this homework specifically, this entails answering the Logarithms and Big-O and Paths questions.**

You should hand in the following files:

- `README.txt`
- `logs.tex`
- `paths.ml`
- `CS17setup.ml`

For this assignment, you must hand in an OCaml file, which must end with extension `.ml`, and a LaTeX-generated `.tex` file. If you are using a departmental Linux system, all your solution files should reside in your `~/course/cs0170/homeworks/hw11` directory.

For this and every assignment, you should also have a `README.txt` file whose first line contains only your CS-department email address, and optionally with a message to the person grading explaining anything peculiar about the handin. For example:

```
README.txt:
jfh@cs.brown.edu
There's nothing to say except that I'm turning in these files plus this README
the way the instructions say that I should.
```

To hand in your solutions to these problems, you must zip your `hw11` directory into a file `hw11.zip`.

Hand in this zip file using the method you learned in the first lab: visit `https://tinyurl.com/cs0170-handin` to get started.

Problems

1 Logarithms and Big-O

Suppose we have procedures whose runtimes H and K are similar:

$$H \in O(n \mapsto \log_2 n) \tag{1}$$
$$K \in O(n \mapsto \log_3 n) \tag{2}$$

How different are they, really? Not different at all!

Task: To prove this, show that

$$n \mapsto \log_2 n \in O(n \mapsto \log_3 n) \tag{3}$$
$$n \mapsto \log_3 n \in O(n \mapsto \log_2 n) \tag{4}$$

To show the first part, exhibit numbers $M, c > 0$ with the property that for every natural number $n > M$, we have

$$\log_2 n \leq c \log_3 n$$

and show that this property holds (using basic logarithm facts and algebra — a few words should suffice).

Then do something similar for the second part. Briefly (in at most a sentence or two) explain why this means that

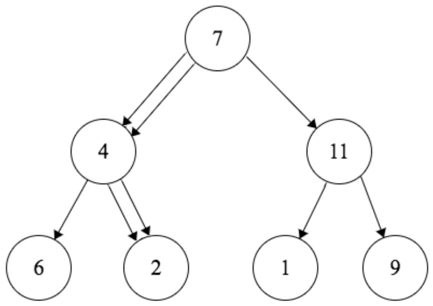
$$\Theta(n \mapsto \log_3 n) = \Theta(n \mapsto \log_2 n).$$

Hint: For the first task, you might want to review the definition and main properties of $\log_a x$. In particular, recall the change of base formula for logs, reproduced below:

$$\log_a x = \frac{\log_b x}{\log_b a}$$

2 Paths

Consider the path from 7 to 2 in the following tree:



We can describe this path using the series of left (L) or right (R) movements that we must make down the tree to get to 2. In this case, we could write the path from 7 to 2 as `[L; R]`, meaning that we first take the left branch from 7, then the right branch from 4.

One way to get from 7 to 2 (in the graph above) is to look for the smaller child of 7 (which is the 4) and then the smaller child of 4 (which is the 2). We can call this a "greedy-minimum" algorithm, because it doesn't find the smallest value in the tree (the 1), but instead just makes, at each node, the "greedy" choice of whichever node is smaller.

For a more general int tree, we need to consider the case where some node has a `Leaf` as one of its children; in that case, we consider the other child to be the "smaller". If a node has *two* `Leaf`s as children, we stop at that node.

Task: Use the following template to write a procedure `gmin` that consumes a nonempty int tree in which all values are distinct and produces a (path, int) pair, where the path is the list of directions followed in the greedy-min algorithm, and the int is the value of the bottom-most node reached.

```
type direction = L | R
type path = direction list
type tree = Leaf | Node of int * tree * tree

let rec gmin (t : tree) : (path * int) = ...
```

For example, if `t` is the tree shown above, then `gmin t` should evaluate to `([L;R], 2)`.

Now let's switch things up a little, and instead of greedy-min, alternate what we do at each step. At the first step, we'll descend to the *larger* of the two children; at the next step, we'll descend to the *smaller* of the two children, and so on, alternating until there are no more non-`Leaf` children. If there's only one non-`Leaf` child, we declare the other child (the `Node` child) to be both the "smaller" *and* the "larger" child. In the example above, this results in visiting the nodes 7, 11, 1.

Task: Write a procedure `amm` that finds the alternating-max-and-min path to a "bottom" node of a nonempty int tree in which all the values are distinct, and returns both the path to that node and the value of that node. If `t` once again denotes the tree in the example, then `amm t` would evaluate to `([R;L],1)`.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS17document by filling out the anonymous feedback form: `http://cs.brown.edu/courses/cs017/feedback`.