CSCM712 – Problem efficiency and solving software March 20th 2026
Cécilia Pradic & Deshan Sumanathilaka

Lab 8: trees, again

For the lab this week, we expect you to put more code in the file `btreeIntro.py` for Tasks 8.1 and 8.2. While the tasks do not formally depend on what you did last week, I strongly suggest you complete Task 7.3 before starting with this lab. Please write tests for your functions before you sign off.

1. **Basic methods** In this question we look at starting to flesh out the class `btree`

   (a) Write the body of the method

   ```
   def depth(self):
   ```

   that returns the depth/height of the tree, i.e. the longest path from the root of the tree to a leaf.

   > **Tip.** *This will be very similar to the* `size` *method which is already implemented in the class. You may want to use th* `max` *function.*

   (b) Consider the following python interaction, assuming `btreeIntro` has been imported with `from btreeIntro import *` beforehand

   ```
   >>> x = btree(5)
   >>> y = btree(5)
   >>> x == x
   True
   >>> x == y
   False
   ```

   Explain why the second comparison evaluated to `False`.

   (c) Write the body of the method

   ```
   def __eq__(self, otherTree):
   ```

   that checks wether the calling object represents the same tree as `otherTree`; it will overload the `==` operator.

   (d) Write the body of the method

   ```
   def prefixTraversal(self):
   ```

   that converts a tree to a list by collecting the labels using a depth-first prefix traversal.

2. **Binary search trees** Now we are going to implement a few methods that work on those `btree`s that are *binary search trees*. A binary search tree

operation should preserve the following invariant that characterize a binary search tree: for every node, the label should be greater than all labels on the left subtree of the node and lesser than the labels in the right subtree if they exist.

(a) Write a method

```
def insert(self, k):
```

that inserts a new number `k` in a binary search tree.

> **Tip.** *If you struggle, feel free to take a look at the code of the following method to get started:*
>
> ```
> def __contains__(self, k):
>     if self.label == k:
>       return True
>     elif self.label < k:
>       if self.left == None:
>         return False
>       else
>         return k in self.left
>     else
>       if self.right == None:
>         return False
>       else
>         return k in self.right
> ```

3. **Pretty-printing please** In this exercise, we will try to write a function that converts a tree into a nice-looking string that pictures a tree. The model for this will be the `tree` command available in most Linux distributions that lists the content of directories in a tree-like format (see Figure 1).

   The file `printtree.py` on canvas contains a class `tree` with a faulty implementation of `__str__` that happens to work exactly on the example `x` defined there, but only on that example. Your task is to re-implement `__str__` correctly so that it may output a sensible string for all possible trees.

   You will note a discrepancy with Figure 1: the characters used make up a slightly uglier picture. This is becasue I restricted myself to ASCII characters for the convenience of some of you: last I checked, the terminal emulator on windows does not support the UTF-8 characters that allows to get a nice picture as in Figure 1. You can test the version with UTF-8 by uncommenting the comments from line 15 to 18. Whichever version you chose, you won't be penalized!

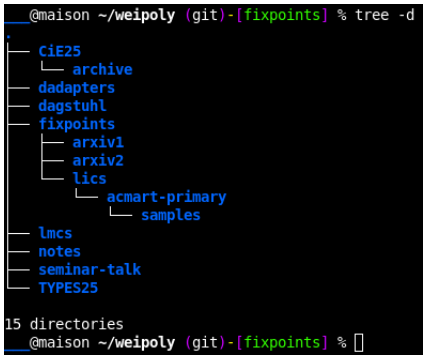

Figure 1: The output of the `tree` command on Cécilia's machine in a folder shared with her PhD student.

> **Tip.** *You may want to:*
>
> - *first compute each output line separately in a list, before joining them using* `'\n'.join(blabla)`
> - *and either have an auxiliary recursive function that allows you to prefix a string by a fixed string passed as an argument, or use the* `map` *function (the latter option is more advanced)*

4. **Optional advanced exercises**

   (a) Write a function

   ```
   def btreeFromString(s):
   ```

   that should convert a string to a tree; this should be the inverse of the method `__str__` that is provided.

   > **Tip.** *Using a stack might be helpful.*

   (b) Write a method

   ```
   def unshareSubtrees():
   ```

   in the class `btree` that removes sharing from a structure with no cycles, but keeps the same underlying unfolding (so the output of the provided `toString` method should remain the same)

   (c) Change the code of the `__str__` method in `btree` so that it prints out useful informations in case there are cycles. You could for instance write the result as some sort of system of recursive equations, as

$$(x)\ 5\ ((2)\ 4\ (x)) \qquad \text{where} \begin{cases} x &= ((5)\ x)\ 7\ y \\ y &= y\ 4\ (x\ 5\ y) \end{cases}$$