Test Assignment

Scala Plugin Internship

Overview

To figure out if we are a good fit for working together, we prepared some tasks for you to evaluate and complete. It's up to you how much of this you do, and how much work you put into it. Of course more complete solutions will be rated higher, but don't spend more than a day on it unless you're having fun.

- Prefer having good working, well-tested, documented code over more features.
- Since the internship focuses on the Scala programming language and ecosystem, the tasks give you a chance to learn a bit about it, if you haven't yet.
- If you find an instruction to be ambiguous, use your judgment on how to solve it.
- You're encouraged to use any resources you find online.

Warmup

Here's a recursive function definition:

- **`f(x) = f(x-1) + f(x-1)`**
- **`f(0) = 1`**

Implement this in Scala.

1. What big-O complexity is your solution?
2. Can you do better? Why / why not / how?

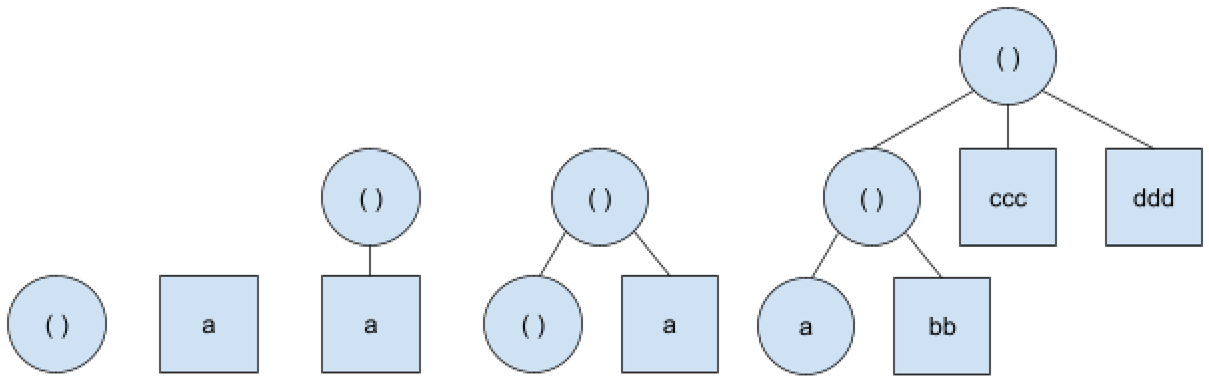
Parsing, Syntax Tree, and Matching

In this task we want to process a simple language *TL* that uses identifiers and parentheses to describe trees.
Trees can either be identifiers (consisting of multiple characters between **`a-z`** and **`A-Z`**) or nodes. Nodes start with opening parentheses and end with closing parentheses. In between they can have further trees. Whitespaces can appear at any position and can be used to separate two identifiers.

The following grammar in EBNF with the start rule **`NODE`** describes the language *TL*:

```
TREE = NODE | ID
NODE = '(' TREE* ')'
ID   = [a-zA-Z0-9]+
```

Here are some examples of trees and representations in *TL*:



`()`   `a`   `(a)`   `(() a)`   `((a bb) ccc ddd)`

Tasks

1. Create a suitable data structure to represent the trees in Scala.
2. Implement a way to check whether two trees are equal. Two trees are equal if they have the same identifier, or when they have the same number of children where all two children with the same index are equal.
3. Implement a method that returns a string by converting a tree into its representation in *TL*.
4. Implement the reverse, i.e. a function, that takes a string, parses it, and returns the tree that was represented by the string in *TL*. Fail and return a message with a reason if the string couldn't be parsed. Parsing a string produced by your method in 3 should return a tree equal to the original one.
5. Now implement a method
   `def replace(tree: Tree, searchTree: Tree, replacement: Tree): Tree`
   It should return a copy of `tree` where all subtrees (including `tree` itself) that are equal to `searchTree` are replaced by `replacement`.

Testing

Test your solutions as thoroughly as possible with one of the many Scala test frameworks.