Mission Sidequest 12.1: Balance your tree

Start date: 22 September 2017
Due: 3 October 2017, 23:59

Readings:

- None

In this sidequest, we will build a balanced binary search tree.

Before we start, let us first recall three basic concepts from the lectures and Mission 11.

- A **tree** of data items are lists whose elements are either data items or trees.
- **Binary trees** are a special kind of trees. A binary tree is either the empty list or a list with three elements. The first element is a binary tree, called its *left subtree*, the second element is a data item called its *value*, in our case a string, and the third element is a binary tree, called its *right subtree*.
- A **binary search tree** is a binary tree that has the following property: all data items in its left subtree are smaller than its value and all data items in its right subtree are bigger than its value (We assume no duplicates here).

In this sidequest, we are going to introduce one new concept, "balanced tree".

- A balanced tree is a tree whose leaves have depths that differ by at most one.

It's time to start. Let's build our own **binary search tree**, hopefully, a balanced one!

Task 1:

Again, recall that a binary tree is either the empty list or a list with three elements. For each list with three elements, we call it a `node` of the tree. Following the tree discipline, define a function `make_node` that takes three parameters, `left_tree`, `number` and `right_tree`. `left_tree` is another binary tree, who is the left subtree of this node; `number` is value stored in this node; `right_tree` is also another binary tree, who is the right subtree of this node.

Please fill in the template given and implement the function `make_node`.

Task 2:

Again, recall that a balanced tree is a tree whose leaves have depths that differ by at most one. Thus, for each node, the size of its left subtree and right subtree should not differ too much. Define a function `make_balanced` that takes a list `xs` as input and build a **balanced binary search tree** using all elements of `xs`. We assume `xs` is already sorted in the ascending order.

Notice: Each node in the **balanced binary search tree** should be built using the `make_node` defined in Task 2. Your marks will be deducted if you break the abstraction.

Hint:

1. For each node, the left subtree and right subtree should be balanced as well. Thus, you may want to call `make_balanced` again to build them.
2. Make use of the `take` and `drop` defined in Lecture 6A to distribute elements into either left subtree, right subtree, or place it as the value of the current node.

Submission

Submit your mission on the Source Academy.

To submit your work to the Source Academy, place your program in the "Source" tab of the online editor within the mission page, save the program by clicking the "Save" button, and click the "Submit" button. Please ensure the required function from each Task is included in your submission. Note that submission is final and that any mistakes in submission requires extra effort from a tutor or the lecturer himself to fix.

IMPORTANT: Make sure you've saved the latest version of your work by clicking the "Save" button before finalizing your submission!