Hands On 1

Angelo Savino

October 8, 2024

Exercise 1

The first exercise requires us to create a method that, given a tree, returns true if it is a bst.

We first define a helper method **is_bst_rec** that takes as input an Option containing a node_id and returns a triple **(is_bst, max, min)**.

The base case of this function is when it receives **None**, in that case we return **(true, u32::MIN, u32::MAX)**, as the empty tree is a BST, and u32::MIN and u32::MAX are respectively smaller and bigger than all numbers that can be represented using 32 bits, which will be useful when combining the results one level higher in the recusrsion.

If we are not in the base case, the function first is recursively called on the left and right child of the node, then the result of the calls is combined in the following way:

- **is_bst** is true if both the left and right subtrees are BSTs **and** if the max value in the left subtree is less or equal than the key in the node of node_id **and** if the min value in the right subtree is greater than the key in the node of node_id, otherwise id set to false.
- max is the maximum value between the max in the subtrees and the key of the current node.
- min is the minimum value between the min in the subtrees and the key of the current node.

The public method **is_bst** calls the helper on the root of the tree, obtaining the triple containing the value of the predicate **is_bst** and the minimum and maximum values contained in the tree. The value of the predicate **is_bst** is returned to the caller.

Exercise 2

The text of the second exercise is the following:

"Given a binary tree in which each node element contains a number. Find the maximum possible path sum from one special node to another special node. Note: Here special node is a node which is connected to exactly one different node."

This is equivalent of finding the best path between two leaves, except for when the root has only one of the two childs. In that case the root is also a *special node* and thus a path from leaf to the root is an admissible solution.

In the same fashion of the first exercise, most of the computation will be done in a private helper and the public function will only be responsible for unwrapping the output and returning the correct value to the caller.

The private helper **max_path_sum_rec** is defined as taking as input an Option containing a node_id and returning a tuple **(bp, bs)**, where **bp** is the maximum path from a leaf to the node with **node_id** and **bs** is the maximum path from leaf to leaf for a given subtree rooted in the node **node_id**. The field **bs** is wrapped in an Option because in the base case (the empty tree) such a path does not exist.

The base case of this function is when it receives **None**, in that case we return the pair **(0, None)**.

If we are not in the base case, the function first is recursively called on the left and right child of the node, then the result of the calls is combined in the following way:

- **bp** is the maximum between the best paths of the subtrees, plus the value of the current node key.
- **bs** is the maximum between the best possible solutions of the two subtrees (if there are any), and the best possible solution passing through the current node, which is given by the sum of the best possible paths in the left and right subtree, plus the current node key.

The public method **max_path_sum** calls the helper on the root of the tree, obtaining the tuple containing **bp**, the best path from the leaf up to the root, and **bs**, the max path from leaf to leaf in the entire tree. The value of the **bs** is unwrapped and then returned to the caller.

Generic implementation

Generic implementation of the solution is available at `https://github.com/AngeloSav/cpc/blob/main/hands_on_1_generic/src/lib.rs`. None of the function logic has been changed to make it work, meaning that the algorithms described also work for signed integers.