Max Over BT Descendants

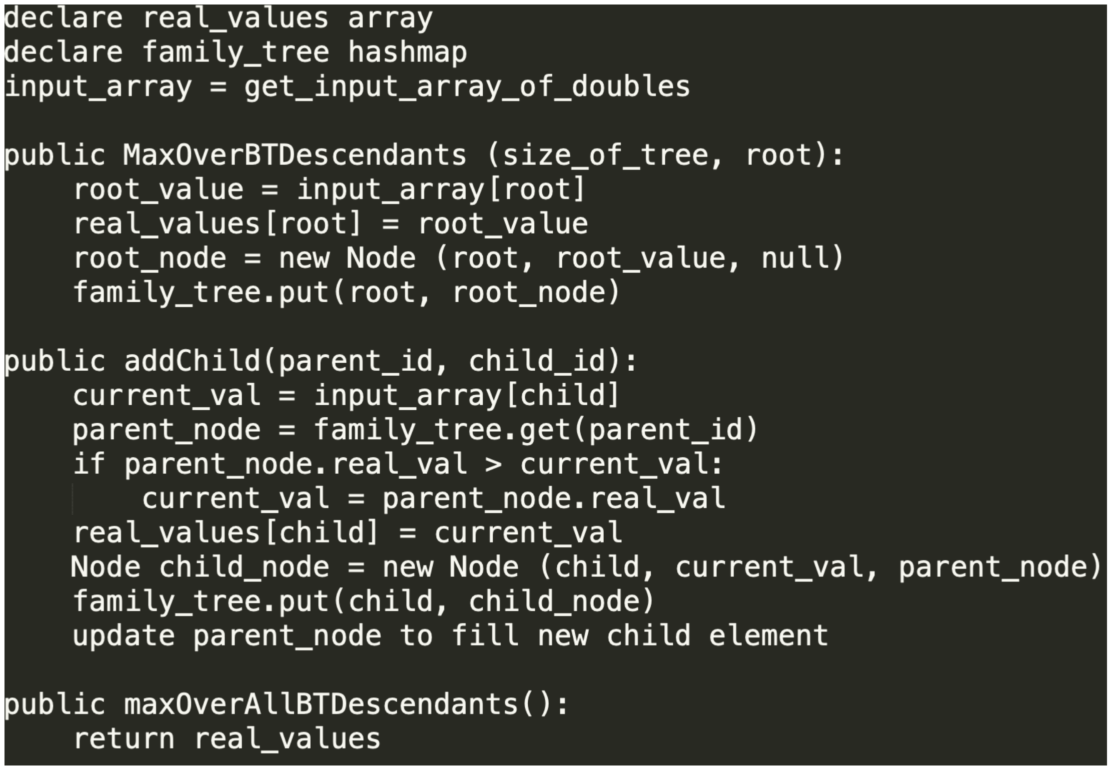
Tom Bohbot

February 2021

1) My algorithm actually beats the linear algorithm time constraint and is a constant time algorithm in respects to the maxOverAllBtDescendants method. My algorithm uses a HashMap as the underlying structure for the binary tree and an inner Node class to store all necessary information. I have a node class which stores the node's key, maximum value, parent node, child one node, and child two node. As my algorithm adds children I calculate the real value of that node on the fly as I check if it is the greatest of all its descendants, and if not then I input its descendant's value. Since I do this every time and child is inputted I only have to check it's parent's real value since that will have the max value of it's ancestral tree. Since my calculations are calculated on the fly I update my array of real values in addChild as well which enables me to simply return the global array of real values whenever someone calls maxOverAllBtDescendants().

2) PseudoCode:

   1) Create an inner Node class that stores the key, real_val, parentNode, firstChildNode, and secondChildNode.



```
declare real_values array
declare family_tree hashmap
input_array = get_input_array_of_doubles

public MaxOverBTDescendants (size_of_tree, root):
    root_value = input_array[root]
    real_values[root] = root_value
    root_node = new Node (root, root_value, null)
    family_tree.put(root, root_node)

public addChild(parent_id, child_id):
    current_val = input_array[child]
    parent_node = family_tree.get(parent_id)
    if parent_node.real_val > current_val:
        current_val = parent_node.real_val
    real_values[child] = current_val
    Node child_node = new Node (child, current_val, parent_node)
    family_tree.put(child, child_node)
    update parent_node to fill new child element

public maxOverAllBTDescendants():
    return real_values
```

4) Correctness Proof:
    a. Direct Proof:
        i. Necessary Prerequisite Knowledge:
            1. The real value of a person is defined by being the greater value between the person's value and the greatest ancestral value of said person.
        ii. The goal of this algorithm is to return the real value of each person inputted into the tree. When the root person is inputted into the tree their value is currently the greatest value in the tree since they are the sole person in the tree. As people are inserted in the tree the algorithm checks if their initial value is greater than their parent's real value. If it is greater, then their initial value becomes their final real value, otherwise their parent's real value becomes the child's final real value, per the definition of a real value. Every child that is added to the tree will compare their value to their parent's value as this is comparing the greatest value of this family to the person's value. One can be sure that the parent's value holds the greatest value in that family since every descendant of the parent checked if it's value was greater than it's respective parent all the way up to the root person. This results in the new child's real value representing the maximum value of its family, which is the definition of a real value. Additionally, since the comparison is made directly to their parent there is no chance that the real value of a different family line can interfere. There is no chance that a diverted family's real value can interfere with said child since the parent node had never made any comparisons to people in the diverted family, as they just made a single comparison to their parent. Once a child has its real value it can simply be added to the ith index of the array returned in maxOverAllBTDescendants. Therefore, whenever a user invokes the method maxOverAllBTDescendants, they can be sure to obtain an array of accurate real values corresponding to a person's value. End of Proof.

5) Performance Proof:
    a. This algorithm runs in linear time in respect to the fact that the algorithm receives n people to add to the tree. The addChild method runs in constant time and receives n calls which causes the algorithm to have a runtime of O(n) or linear. The method itself is constant time because the underlying datastructure of the method is a HashMap and all methods used in addChild have an amortized O(1) runtime. In the addChild method I calculate the array of real values on the fly which enables me to simply return a global array whenever maxOverAllBTDescendats is invoked which is also just a constant time operation. Therefore, my maxOverAllBTDescendats method runs in constant time and my addChild method runs in linear time due to the fact that it will receive n invocations.