NTIN066 - solutions 2

David Mareček

February 23, 2022

1. What is the worst-case complexity of the SUCC function from your first assignment?

$O(n)$ for a generally unbalanced tree, $O(log(n))$ for a balanced version

2. What is the average complexity of SUCC across all the keys in a tree?
Consider enumeration of all keys in a binary search tree using MIN and n times SUCC. Prove that although a SUCC requires $\Theta(\log(n))$ time in the worst case, the whole enumeration takes only $\Theta(n)$ time. Can we formulate any amortization argument?

Hint: Each edge is used exactly two-times.

3. Show how to make a BST perfectly balanced in $\Omega(n)$ time.
How to manage it with as little additional memory as possible? It is easy to prove it for $O(n)$, harder for $O(\log(n))$, but it is also possible to use only $O(1)$ additional memory.

Hint: https://ksp.mff.cuni.cz/h/ulohy/26/reseni2.htmltask-26-2-5
Logarithmic memory: Using rotations, create a linked list from the tree. Then, use a recursive function, which gets the pointer to the list and a number k and returns balanced tree of the first k items and the pointer to the rest of the list.

4. Show that either Insert or Delete in a perfectly balanced BST tree must have worst-case time complexity $\Omega(n)$.
This is easy for perfectly balanced trees on $n = 2^k - 1$ vertices, whose shape is uniquely determined.

The shape of the tree with $n = 2^k - 1$ vertices is uniquely determined: the middle item must be in the root, and the same holds recursively for the subtrees. If we number the nodes by 1 to n, we see that all odd nodes are leaves. Let's perform two operations Delete(1) and Insert($n+1$). The tree still has $n = 2^k - 1$ vertices, but with the even nodes as leaves now. Therefore, the Insert operation changes all the nodes in the tree and its complexity is $O(n)$. If we swap these two operations, then the complexity of Delete is $O(n)$.

5. Add a Delete operation into lazily balanced trees
Use the same Delete as iplemented in ordinary BSTs. Use the same potential to analyze it.

Consider the three possibilities: a) deleted node is a leaf, b) deleted node has just one child, c) deleted node has two children. In all three cases, you need $O(log(n))$ to perform the operation. The size of all nodes on the path to the root decreases by 1. The contributions of these vertices will therefore increase by at most 2 (they will usually change by exactly one, but because of the clamping, it can jump between 0 and 2). So we increase the potential at most by another $O(log(n))$, so the total amortized cost is $O(log(n))$. When rebuilding a subtree rooted by v, its potential is $\Phi(v) > 1/3 \cdot s(v)$, which is enough for building the ballanced subtree (in linear time).

6. What would go wrong if we forgot to add the exception for difference 1 in the definition of the potential $\Phi(v)$ in lazily balanced trees?

When rebuilding the subtree, the decrease of the potential could be less then $1/3 \cdot s(v)$, because even a balanced tree could have some non-zero potential.

7. Decrement in Binary counter
What goes wrong with the amortized complexity if we also want to decrement? Could we improve the counter so that the amortized complexity is $O(1)$ for each possible sequence of increments and decrements?

Please, read the solution here:
https://courses.engr.illinois.edu/cs473/sp2009/notes/11-amortize.pdf