CSCI-GA.1170-001/002 Fundamental Algorithms October 29, 2015

Solutions to Problem 4 of Homework 5 (14 points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, October 13*

Assume that you are given a 2-3 tree T containing n distinct elements.

(a) (4 points) Show how to find the successor of a given element $x \in T$ in time $O(\log n)$.

Solution:

The leaves of 2-3 Trees are in a sorted order when seen from left to the right. Therefore the successor of any given node is the adjacent leaf node to it's right. In order to find the successor of any leaf node x, we proceed as follows.

- Go up the tree to the first ancestor of x that has a child to the right. (i.e we find the first ancestor of x, such that the sub tree in which x is located has a right sibling)
- Find the leftmost leaf node in this sibling sub tree by following the left children all way until the leaf. This leaf we find is the successor of x

In the worst case, we start from x and go up the tree until the root and then till the bottom to the minimal leaf node in root's right subtree. Therefore, the worst case running time is $O(\log n)$

□

(b) (4 points) Show that if the input element x is chosen *uniformly at random* from T, then your procedure from part (a) runs in *expected* time $O(1)$.

Solution:

Consider a node v which has three children a, b and c. Therefore, $succ(a) = b$ and $succ(b) = c$. Finding the successors of these two nodes takes only $O(1)$ time and to find $succ(c)$ we need to go up the tree until we find an ancestor such that it has a right child. This might take $O(\log n)$ time. Therefore, the probability that the successor operation takes only $O(1)$ time is $2/3$ and $O(\log n)$ is $1/3$.

If the node v only two children a and b then it is clear that the probability that the successor operation takes only $O(1)$ time is $1/2$ and $O(\log n)$ is $1/2$. Therefore, in the worst-case scenario, every node of the tree has only 2 children i.e a perfectly balanced binary tree

Let n be the total number of leaf nodes and h be the height of the 2-3 Tree i.e $h = \log n$

The number of leaf nodes x such that ancestor of x has a right child at height 1 from x is $n/2$. Therefore, finding successor of these $n/2$ nodes takes time $n/2.O(1)$

The number of leaf nodes x such that ancestor of x has a right child at height 2 from x is $n/4$. Therefore, finding successor of these $n/4$ nodes takes time $n/4.O(2)$

The number of leaf nodes x such that ancestor of x has a right child at height 3 from x is $n/8$. Therefore, finding successor of these $n/4$ nodes takes time $n/8.O(3)$

$\vdots$

The number of leaf nodes x such that ancestor of x has a right child at height $\log n$ from x is 1. Therefore, finding successor of this 1 node takes time $O(\log n)$

Let $T(n)$ be the expected running time, then

$$\begin{aligned} T(n) &= \frac{1}{n}[\frac{n}{2}O(1) + \frac{n}{4}O(2) + \frac{n}{8}O(3) + \ldots + 1.O(\log n)] \\ &= \frac{1}{n}\sum_{i=1}^{\log n}\frac{n}{2^i}O(i) \\ &= \sum_{i=1}^{\log n}\frac{O(i)}{2^i} \end{aligned}$$

Consider the summation, $\sum_{i=0}^{\infty} x^j = \frac{1}{1-x}$

Take derivative on both sides and multiply by x we get $\sum_{i=0}^{\infty} ix^i = \frac{x}{(1-x)^2}$

Put $x = 1/2$ in the above equation, we get $\sum_{i=0}^{\infty}\frac{i}{2^i} = 2 \implies \sum_{i=0}^{\infty}\frac{O(i)}{2^i} = O(1)$

$$\begin{aligned} T(n) &= \sum_{i=1}^{\log n}\frac{O(i)}{2^i} \leq \sum_{i=0}^{\infty}\frac{O(i)}{2^i} \\ \therefore T(n) &= O(1) \end{aligned}$$

□

Assume that we wish to augment our 2-3 tree data structure so that that each node v maintains a pointer $v.succ$ to the successor of v, so that queries for the successor of an element can be answered in $O(1)$ time *worst-case*.

(c) (6 points) Show that the 2-3 trees can be augmented while maintaining $v.succ$, such that the INSERT and DELETE operations can still be performed in $O(\log n)$ time. (**Hint**: Think of a linked list.)

Solution:

Given that leaf node v has a pointer $v.succ$ to the successor of v, so the leaf nodes of the 2-3 Tree form a sorted liked list.

INSERT

Let y be the node to be inserted into the tree. Perform the usual Insert operation in $O(\log n)$ time.

- If y is the leftmost leaf node in the tree. Let z be the adjacent leaf node of y to it's right, then $y.succ = z$. This takes $O(1)$ time
- If y is the rightmost leaf node in the tree. Let z be the adjacent lead node to y to it's left, then $z.succ = y$ and $y.succ = null$. This takes $O(1)$ time
- Else, Let x be the predecessor of y and let z be $x.succ$ then $x.succ = y$ and $y.succ = z$. This step takes $O(\log n)$ time

Therefore, the Insert operation still takes $O(\log n)$ time while maintaining the *succ* field

DELETE

Let y be the node to be deleted in the tree

- If y is the leftmost or the rightmost leaf node to be deleted in the tree, then perform the usual Delete operation. This step takes $O(\log n)$ time
- Else, Let x be the predecessor of y and $y.succ$ be z, then perform the usual Delete operation and update $x.succ$ to z. This step takes $O(\log n)$ time

Therefore, the Delete operation still takes $O(\log n)$ time while maintaining the *succ* field $\square$