Homework 2 Solutions
Fundamental Algorithms, Fall 2005, Professor Yap

Due: Wed Oct 19, in class.
SOLUTION PREPARED BY Instructor and T.A.s

INSTRUCTIONS:

- Please read questions carefully. When in doubt, please ask.
- In view of the midterm, we will post the solution to this problem set on Wed evening of due date. Absolutely no late homework is entertained.

Question 1

(20 Points)
(a) Here is the list of keys produced by a post-order traversal of a binary search tree:

$$2, 1, 4, 3, 6, 7, 9, 11, 10, 8, 5, 13, 16, 15, 14, 12$$

Draw this binary search tree.
(b) Describe a general algorithm to reconstruct a BST from its post-order traversal list of keys (as in part (a)). HINT: Recursively, of course.
ANSWER:
(a) Figure 1 shows the corresponding binary search tree.

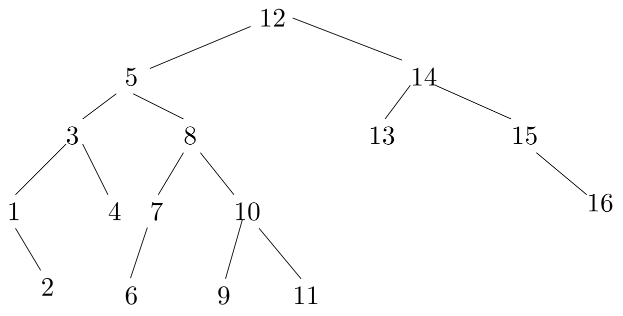

Figure 1: Binary Search tree

(b) In general, suppose $(k_1, \ldots, k_n)$ are the post-order keys. Since this is post-order, we know that k_n is the root. Then we can get two subsequences from $(k_1, \ldots, k_{n-1})$ (note that the elements appear in the same order as the original sequence) comprising (1) all the nodes that are smaller than k_n, and (2) all the nodes that are greater than k_n. Recursively, we can construct the BST corresponding to (1) and the BST corresponding to (2). These two BST's are just the left and right subtree of the root k_n. The base case is trivial, namely when there is only one element in the post-order traversal.
COMMENTS:
For part (b), many students did not realize that one has to just search for the first index in the list where the element is smaller then the root (which is the last elemet of the list). Given this index, one can easily split the list into two parts as suggested in the solution.

Question 2

(10 Points)
Give an algorithm to check whether a binary search tree T is an AVL tree. Your algorithm should take time $O(|T|)$. HINT: Use one of the tree-traversal methods as a shell for this algorithm.

ANSWER:

We can use a post-order traversal shell. Assume that $post(v)$ returns the height of the subtree at v. The height of the tree is ∞ if it is not AVL. Then the algorithm is as follows:

```
POST($v$)
    If $v = \mathsf{nil}$, return($-1$)
    $h_L = post(v.Left)$
    $h_R = post(v.Right)$
    If ($|h_L - h_R| > 1$ or $h_L + h_R = \infty$) return($\infty$)
    Else return($1 + \max\{h_L, h_R\}$)
```

COMMENTS:

(1) If you forgot the base case, i.e. when the tree is an empty tree, then a point has been deducted.

(2) Some of the solutions were using the depth of a node, the distance from the root to the node, rather than the height, the distance from the node to its most distant leaf. Four points have been deducted for this mistake.

Question 3

(5 Points)
TRUE or FALSE: Justification is always needed in such questions.
Recall that a rotation can be implemented with 6 pointer assignments. Suppose a binary search tree maintains successor and predecessor links (denoted u.`succ` and u.`pred` in the text). This is in addition to the standard links to the children and parent. Now rotation requires 12 pointer assignments.

SOLUTION:

FALSE. Rotation does not affect the successor and predecessor links of a binary search tree. (That is because the inorder traversal of a binary tree is unaffected by rotation.) So 6 assignments suffice.

COMMENTS:

(1) Most of the answers were correct even though the jusification was wrong; in this case 2 points were deducted. More precisely, some of you said that it will not take 12 assignments but 7, or some other number. This is not correct.

Question 4

(5+5+10+10 Points)
Suppose we define the height of the empty tree to be $-\infty$ in a new definition of 'AVL' trees, but everything else remain the same as before. We want to compare the original AVL trees with this new 'AVL' trees.

(a) TRUE or FALSE: every 'AVL' tree is an AVL tree.

(b) Let $\mu'(h)$ be defined (similar to $\mu(h)$ in the Lecture Notes) as the minumum number of nodes in an 'AVL' tree. Determine $\mu'(h)$ for all $h \leq 5$.

(c) Show a relationship among $\mu'(h), \mu(h)$ and $F(h)$ where $F(h)$ is the standard Fibonacci numbers.

(d) Give a good upper bound on $\mu'(h)$.

ANSWER:

(a) The 'AVL' trees of heights 0 and 1 are both unique, and they are AVL. Inductively, all other 'AVL' trees of greater heights are built from these two 'AVL' trees, using the rule about height balance. Since these are exactly the same rules as for regular AVL trees, this proves that all 'AVL' trees are AVL.

The converse does not hold: The tree having just two nodes, i.e., a parent and its child is not 'AVL' even though it is AVL. It is not 'AVL' because the difference between the height of the two children is $|-\infty - 1| > 1$. We also have the convention $(-\infty) - (-\infty) = 0$ (mathematically, this is undefined).

(b) $\mu'(0) = 1, \mu'(1) = 3, \mu'(2) = 5, \mu'(3) = 9, \mu'(4) = 15, \mu'(5) = 25$.

(c) We claim that $\mu'(h) = \mu(h) + F(h)$. The proof is by induction on h. The claim is true in the base case

$h = 0$ since $\mu'(0) = \mu(0) = 1$ and $F(0) = 0$. Suppose that the claim is true for h, i.e.

$$\mu'(h) = \mu(h) + F(h). \tag{1}$$

Then we know

$$\begin{aligned}\mu'(h+1) &= \mu'(h) + \mu'(h-1) + 1 \\ &= \mu(h) + F(h) + \mu(h-1) + F(h-1) + 1 \quad \text{(from (1))} \\ &= \mu(h+1) + F(h),\end{aligned}$$

since for AVL trees we know that 1) $\mu(h+1) = \mu(h) + \mu(h-1) + 1$ and 2) the Fibonacci numbers satisfy the recurrence $F(h+1) = F(h) + F(h-1)$.

(d) We know that $F(h) = \phi^h$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. We also know that $\mu(h) \leq 2^h$. Thus from (c) above we get that $\mu'(h) \leq \phi^h + 2^h$. However, we can show something better: $\mu'(h) \leq 2^h - 1$ for $h \geq 4$. Again we prove using induction on h. The base case $h = 4$ follows from (b). Assuming the claim to be true for h we have

$$\begin{aligned}\mu'(h+1) &= \mu'(h) + \mu'(h-1) + 1 \\ &\leq 2^h - 1 + 2^{h-1} \\ &< 2^{h+1} - 1.\end{aligned}$$

A better bound is: $\mu'(h) \leq 3\phi^h - 1$. The base case for $h = 0$ and $h = 1$ is clearly true. Inductively we have

$$\begin{aligned}\mu'(h+1) &= \mu'(h) + \mu'(h-1) + 1 \\ &\leq 3\phi^h + 3\phi^{h-1} - 1 \\ &= 3\phi^{h-1}(\phi + 1) - 1 \\ &= 3\phi^{h+1} - 1.\end{aligned}$$

COMMENTS:

1) For (c), 5 points are given by default; 8 if the relationships are correct but without or with an incorrect proof; otherwise, full points.

2) For (d), 5 points are awarded if a correct bound, even though with a wrong argument, is provided; otherwise, full points.

3) For (d), most of the students made statements saying $\mu'(h) = O(C^h)$ after giving the inequality $\mu'(h) \leq KC^h$, for some constant K. Please note that the inequality is a stronger and more precise statement then the Big-O notation and hence it is sufficient to just give the inequality.

Question 5

(15 Points)

Inserting the following keys $0, 3, 6, 9, 12, 15, 18, 21$ (in this order) into an initially empty AVL tree. Further insert the keys $10, 11, 13, 14, 16, 17$ (in this order) into the same tree. Show the resulting tree at the end of each insertion (you may skip those that caused no rotations).

To delete a node u with two children, assume that you move the successor of u into the place of u, and then Cut the node where the successor comes from.

Now, delete 9 and then 6. Show the result after each single rotation or double rotation.

ANSWER:

See Figure 2.

COMMENTS: 10 points are given if the tree given was partially correct and 13 if insertion steps were correct, but not deletion.

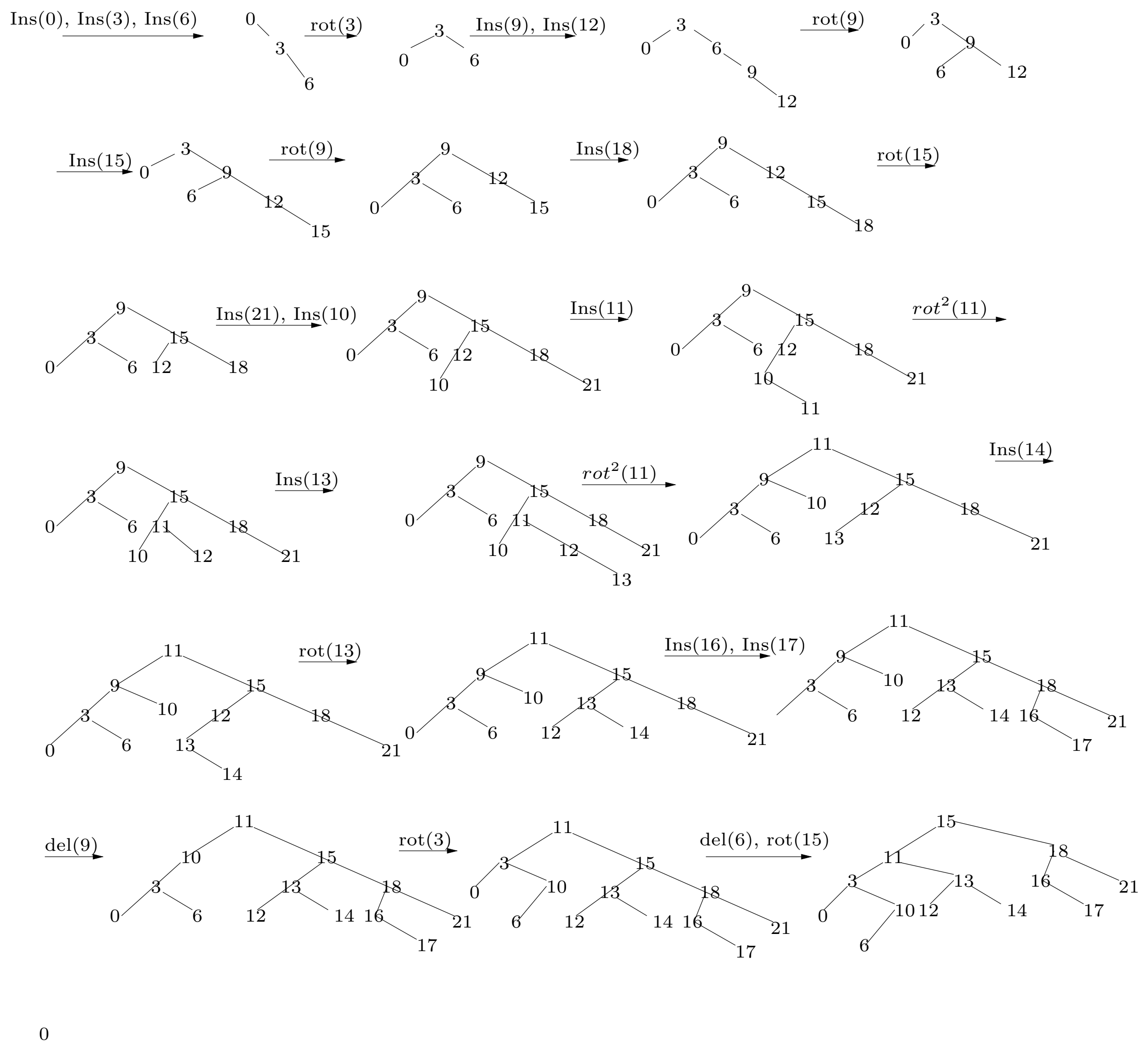

Figure 2:

Question 6

(40 Points)
We want to implement (a, b, c)-trees with the parameter $c = 2$. The nodes of the search tree are stored on the disk. The root is assumed to be always in main memory. To transfer data between disk and main memory, we assume a UNIX-like environment where memory blocks have size of 512 bytes. The reading or writing of one memory block constitute one disk access. Each disk I/O is two or three orders of magnitude slower than each CPU operation. Assume that each pointer is 4 bytes and each key 6 bytes.
(a) Choose a and b optimally.
(b) Suppose we have a billion items that is stored in our $(a, b, 2)$-tree. How many disk accesses is needed to lookup an item? To insert an item? To delete an item?
(c) Assume that the root is always resident in main memory. Give upper and lower bounds on the number of blocks needed to store the internal nodes of this tree.
(d) Suppose that each data is 8 bytes (it is probably only a pointer). Choose a', b' optimally. Also give upper and lower bounds on the number of blocks needed to store the leaves of the tree.

ANSWER: (a) Since in any node the number of keys are at most $b - 1$ and the number of pointers b, we want $6(b - 1) + 4b \leq 512$ to maximize the utilization in any given node; this yields $b = 51$, since b is an integer. The parameter a should satisfy $a = \left\lfloor \frac{cb+1}{c+1} \right\rfloor$; substituting $b = 51$ and $c = 2$ yields $a = 34$.

(b) It is clear that the search tree, containing billion items, whose the root has exactly 2 children and all other nodes have exactly a children has the maximum depth and hence the longest lookup time. The height h of this search tree is bounded by $1 + \lfloor \log_a \lfloor n/2 \rfloor \rfloor$, where $a = 34$ and $n = 10^9$; thus height is bounded by 6. Since at each lookup we need one disk access the number of disk accesses for looking up an item is bounded by the height 6. In case of inserting an item we might have to perform a split or merge. In case of a split we might need to lookup at $c-1$ of the neighbours of a node, thus requiring $c-1$ disk accesses, and when we split we have to write $c+1$ new memory blocks; thus at any given depth we have to perform at most $2c+1$ disk accesses, the extra one coming from the fact that we have to bring the node itself into memory first. This gives us the disk accesses to insert an item as $(2c+1)h$; to this we also have to add the number of initial lookups that are at most h and 2 additional disk accesses resulting from the splitting of the root. Thus the total disk accesses to insert an item is bounded by $(2c+2)h + 2$; with our values this gives us 38.

In case of deleting an item from the tree we do not have to do the two additional disk accesses resulting from the splitting of the root thus giving us a bound of $2h(c+1)$ i.e. 36.

(c) From lecture notes we know that the height h of the tree satisfies

$$\lceil \log_b n \rceil \le h \le 1 + \lfloor \log_a \lfloor n/2 \rfloor \rfloor .$$

When $n = 10^9$, $a = 34$, and $b = 51$ the lower and upper bound coincide and are 6. The minimum number of blocks needed to store n items are $\sum_{i=1}^{h} b^i$ since the root always resides in the memory. Similarly, the maximum number of nodes needed to store n items are $2\sum_{i=1}^{h} a^{i-1}$.

(d) Since the leaves store both the key and the associated data, and each such pair occupies 10 bytes, b' satisfies $10b' \le 512$ thus implying that $b' = 36$; since $a' \le \frac{cb+1}{c+1}$ we have $a' = 24$. Thus the number of blocks N needed to store the leaves satisfies

$$\left\lceil \frac{n}{b'} \right\rceil \le N \le \left\lfloor \frac{n}{a'} \right\rfloor .$$

COMMENTS: Each part was worth 10 points.

(1) For (a), 7 points were awarded if one of a or b was wrong, and 5 if both were wrong.

(2) For (b), the breakup was 2 points for lookup and 4 each for insertion and deletion. Full points were given if you roughly mentioned the number of disk accesses accrued due to splitting or merging; a point was deducted if you forgot to do the accounting for all the levels of the tree even afer giving the all most correct answer for each level.

(3) For (c), each bound was worth 5 points. Full points were given if you showed the correct summation for getting the bound on the nodes. For the case of an upper bound, if you forgot to metion that the root has at least 2 children (and $2a^2$, $2a^3$ so on) then a point was deducted.

(4) For (d), if one of the values of a' or b' was wrong then 7-8 points were given; if the bounds were missing, or incorrect, even though the above values were correct, then 2-3 points were deducted.

Question 7

(10 Points)
The root of an (a, b)-tree has between 2 to b children. Suppose we want the root, if non-leaf, to have between a and b children, just like other nodes. Here is the idea: let us allow the root, when it is a leaf, to have between 0 and $a'a - 1$ items. As usual (a', b') are the bounds on items in leaves. When this limit is exceeded, the root has $a'a$ items and we can split it into a leaves, each with a' items. The beauty is that there is no exception with respect to the (a, b) bound. Discuss the issues that might arise with this design.

SOLUTION Unfortunately, there is trouble in deletion. After a deletion, suppose we want to do a generalized merge on the children of the root. If the root has only a children, then this is not allowed. (Previously, we had no problems, as the root was allowed to have less than a children.) In this case, we know that these a children of the root has exactly $a^2 - 1$ children among them. We must then merge all of them into a single root with $a^2 - 1$ children. With this additional condition, we can now achieve our original goal of not having less than a children. But unless $b \ge a^2 - 1$, we still have an exception at the root – we must allow the root to have between a and $a^2 - 1$ children. Are we better off than the previous exception at the root? I think that in practice, this is better than the standard exception.

COMMENTS: Full points were awarded if you mentioned the problem arising out of splitting at the root; 6 points, if there was no such mention at all.