Problem Set 3 Solutions

Reading: Chapters 12.1-12.4, 13, 18.1-18.3

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered in the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

Three-hole punch your paper on submissions.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct algorithms which are *which are described clearly*. Convoluted and obtuse descriptions will receive low marks.

Exercise 3-1. Do Exercise 12.1-2 on page 256 in CLRS.

Exercise 3-2. Do Exercise 12.2-1 on page 259 in CLRS.

Exercise 3-3. Do Exercise 12.3-3 on page 264 in CLRS.

Exercise 3-4. Do Exercise 13.2-1 on page 278 in CLRS.

Problem 3-1. Packing Boxes

The computer science department makes a move to a new building offering the faculty and graduate students boxes, crates and other containers. Prof. Potemkin, afraid of his questionable tenure case, spends all of his time doing research and absentmindedly forgets about the move until the last minute. His secretary advises him to use the only remaining boxes, which have capacity exactly *1 kg*. His belongings consists of n books that weigh between 0 and 1 kilograms. He wants to minimize the total number of used boxes.

Prof. Potemkin realizes that this packing problem is NP-hard, which means that the research community has not yet found a polynomial time algorithm[1] that solves this problem *exactly*.

He thinks of the *heuristic approach* called BEST-PACK:

1. Take the books in the **order in which they appear** on his shelves.
2. For each book, scan the boxes in **increasing order of the remaining capacity** and place the book in the first box in which it fits.

(a) Describe a data structure that supports efficient implementation of BEST-PACK. Show how to use your data structure to get that implementation.

Solution: BEST-PACK can be implemented using any data structure that supports the following three operations:

1. Insert(x), where x is an element and key$[x]$ is a number
2. Delete(x)
3. Successor(x), which reports the smallest x such that key$[x] \geq k$

There are several ways to obtain such a data structure. For example, one can use red-black trees or $2-3$ trees. Because they are balanced, they support Insert, Delete and Successor operations in $O(\log n)$ time. Even though the Successor operation was not explained for $2-3$ trees, they can be implemented by modifying search.

Our implementation is as follows: We use the remaining capacity of the boxes as the key in the binary tree. Suppose that the elements weigh $w_1, \ldots, w_n$. Then, for a given book with weight w_i, if there are no boxes that are already used and whose remaining capacity is greater than w_i (i.e., the successor of w_i), then we assign w_i to a new box.

(b) Analyze the running time of your implementation.

Solution: The BEST-PACK implementation performs $O(n)$ operations on the data structure which implies that the total running time is $O(n \log n)$

[1]That is, an algorithm with running time $O(n^k)$ for some fixed k.

Soon, Prof. Potemkin comes up with another heuristic WORST-PACK, which is as follows:

1. Take the books in the order in which they appear on his shelves.
2. For each book, find a partially used box which has the **maximum** remaining capacity. If possible, place the book in that box. Otherwise, put the book into a new box.

(c) Describe a data structure that supports an efficient implementation of WORST-PACK. Show how to use your data structure to get that implementation.

Solution: WORST-PACK can be implemented using any priority queue data structure. We learned in recitation that a heap implements this data structure in $O(\log n)$ time. You can also use a balanced search tree to implement these operations.

Our implementation is as follows: Pick a book. Delete the maximum from the priority queue. If the capacity is greater than the weight of the book, insert the book and reduce the capacity of the box. Reinsert the box in the priority queue. Otherwise pick a new box and insert the book.

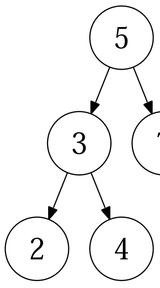
(d) Analyze the running time of your implementation.

Solution: Our implementation performs $O(n)$ operations. This means that the total running time is $O(n \log n)$.

Problem 3-2. AVL Trees

An AVL tree is a binary search tree with one additional structural constraint: For any of its internal nodes, the height difference between its left and right subtree is at most one. We call this property *balance*. Remember that the height is the maximum length of a path to the root.

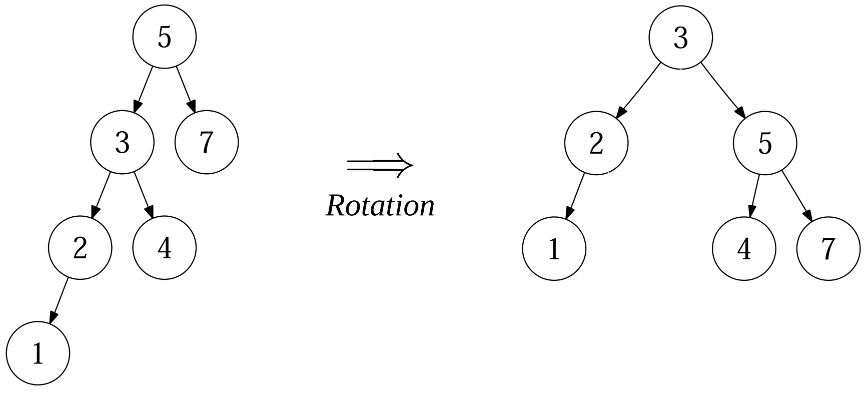
For example, the following binary search tree is an AVL tree:



Balanced AVL Tree

Nevertheless, if you insert 1, the tree becomes unbalanced.

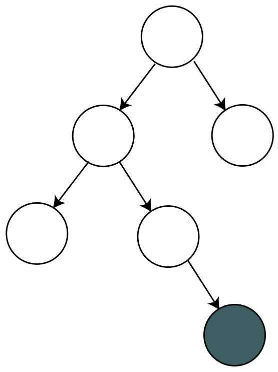
In this case, we can rebalance the tree by doing a simple operation, called a rotation, as follows:



See CLRS, p. 278 for the formal definition of rotations.

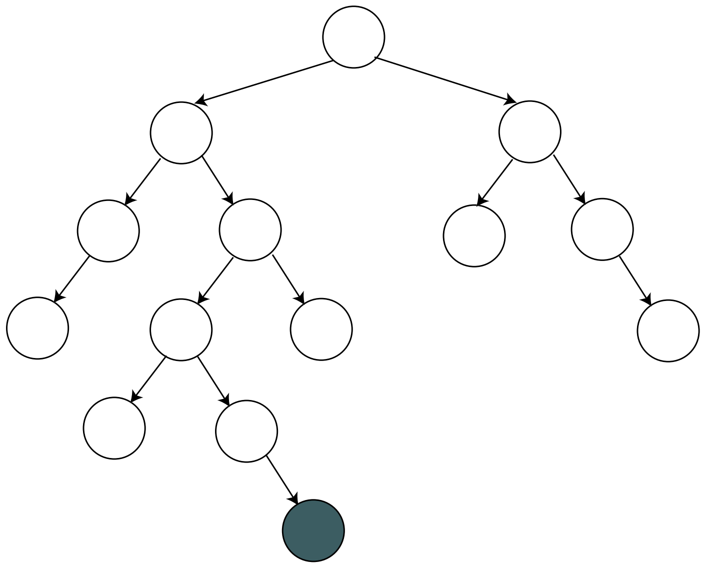
(a) If we insert a new element into an AVL tree of height 4, is one rotation sufficient to re-establish balance? Justify your answer.

Solution: No, one rotation is not always sufficient to re-establish balance. For example, consider the insertion of the shaded node in the following AVL tree:



Though the original tree was balanced, more than one rotation is needed to restore balance following the insertion. This can be seen by an exhaustive enumeration of the rotation possibilities.

The problem asks for a tree of height 4, so we can extend the above example into a larger tree:



(b) Denote the minimum number of nodes of an AVL tree of height h by $M(h)$. A tree of height 0 has one node, so $M(0) = 1$. What is $M(1)$? Give a recurrence for $M(h)$. Show that $M(h)$ is at least F_h, where F_h is the hth Fibonacci number.

Solution: $M(1) = 2$. For $h \geq 2$, the tree will consist of a root plus two subtrees. Since the tree is of height h, one of the subtrees must be of height $h-1$. The minimum number of nodes in this subtree is $M(h-1)$. Since the height of the subtrees can differ by at most 1, the minimum number of nodes in the other subtree is $M(h-2)$. Thus the total number of nodes is $M(h) = M(h-1) + M(h-2) + 1$.

Note that $M(h)$ is remarkably similar to the Fibonacci numbers and that the recursion holds for the worse case AVL trees, which are called Fibonacci trees. It is easy to show by induction that $M(h) = F(h+3) - 1$. Note that, as shown in Problem Set 1, $F(h) \geq \frac{\phi^h}{\sqrt{5}} - 1$ where $\phi = \frac{1+\sqrt{5}}{2}$. This implies that $M(h) \geq \frac{\phi^{h+3}}{\sqrt{5}} - 2$.

(c) Denote by n the number of nodes in an AVL tree. Note that $n \geq M(h)$. Give an upper bound for the height of an AVL tree as a function of n.

Solution: We know that $n \geq M(h) \geq \left(\frac{\phi^{h+3}}{\sqrt{5}} - 2\right)$. Therefore, solving for h, we get that h is $O(\lg n)$.