Design of Algorithms - Homework III (Solutions)

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Problems

1. Professor Krustowski claims to have discovered a new sorting algorithm. Given an array $\mathbf{A}$ of n numbers, his algorithm breaks the array into 3 equal parts of size $\frac{n}{3}$, viz., the first third, the middle third and the bottom third. It then recursively sorts the first two-thirds of the array, the bottom two-thirds of the array and finally the first two-thirds of the array again. Using mathematical induction, prove that the Professor has indeed discovered a correct sorting algorithm. You may assume the following: The input size n is always some multiple of 3; additionally, the algorithm sorts by brute-force, when n is exactly 3. Formulate a recurrence relation to describe the complexity of Professor Krustowski's algorithm and obtain tight asymptotic bounds.

   Solution: As per the specifications, the algorithm works correctly, when $n \leq 3$.

   Assume that the algorithm works correctly, as long as $1 \leq n \leq k$, where $k > 3$.

   Now consider the case where the array $\mathbf{A}$ contains $k+1$ elements.

   We divide the array $\mathbf{A}[1 \cdot\cdot k+1]$ (conceptually) into the following regions: $L_1 : \mathbf{A}[1 \cdot\cdot \frac{k+1}{3}]$, $L_2 : \mathbf{A}[\frac{k+1}{3}+1 \cdot\cdot \frac{2}{3} \cdot (k+1)]$ and $L_3 : \mathbf{A}[\frac{2}{3} \cdot (k+1) + 1 \cdot\cdot k+1]$. Thus, the first recursive invocation is called on $L_1 \cup L_2$, the second recursive invocation is called on $L_2 \cup L_3$, and the third recursive invocation is called on $L_1 \cup L_2$ again.

   By the inductive hypothesis, the first recursive invocation will return a correctly sorted array, since the size of the input array is at most k. (Note that $\frac{2}{3} \cdot (k+1) \leq k$ as long as $k \geq 2$.) It follows that after the first recursive invocation, each element in L_2 is at least as large as every element in L_1. Arguing similarly, we observe that the second recursive invocation correctly sorts the set $L_2 \cup L_3$; further every element in L_3 is at least as large as each element in L_2. From the correctness of the first recursive invocation, we can therefore conclude that each element in L_3 is at least as large as every element in $L_1 \cup L_2$; further as per the inductive hypothesis, all the elements in L_3 have been correctly sorted. The third recursive invocation completes the sorting procedure, since by the inductive hypothesis, it correctly sorts $L_1 \cup L_2$ and thus the set $L_1 \cup L_2 \cup L_3$ is correctly sorted.

   Let $T(n)$ denote the running time of Professor Krustowski's algorithm. We have,

$$\begin{aligned} T(n) &= 3, \ n \leq 3 \\ &= 3 \cdot T\Big(\frac{2}{3} \cdot n\Big), \ \text{otherwise} \end{aligned}$$

   The recurrence relation fits the template of the Master Theorem, with $a = 3$, $b = \frac{3}{2}$ and $f(n) = 0$.

   It is easy to see that $f(n) = O(n^{(\log_{1.5} 3)-\epsilon})$ for some $\epsilon > 0$, so $T(n) \in \Theta(n^{2.7})$ by the master theorem (case 1).

   □

2. Assume that you are given a chain of matrices $\langle A_1 \ A_2 \ A_3 \ A_4 \rangle$, with dimensions 2×5, 5×4, 4×2, and 2×4, respectively. Compute the optimal number of multiplications required to calculate the chain product and also indicate what the optimal order of multiplication should be using parentheses.

Solution: Let $m[i,j]$ denote the optimal number of multiplications to multiply the chain $\langle A_i, A_{i+1}, \ldots, A_j \rangle$, where matrix A_i has dimensions $d_{i-1} \times d_i$. As per the discussion in class, we know that

$$\begin{aligned} m[i,j] &= 0, \text{ if j} \leq \text{i} \\ &= \min_{k:i \leq k < j} m[i,k] + m[k+1,j] + d_{i-1} \cdot d_k \cdot d_j \end{aligned}$$

Computing $\mathbf{M} = [m[i,j]], i = 1,2,3,4;\ j = i, i+1, \ldots, 4$, in bottom-up fashion, we get,

$$\mathbf{M} = \begin{bmatrix} 0 & 40 & 56 & 72 \\ 0 & 0 & 40 & 80 \\ 0 & 0 & 0 & 32 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

By the above table, the optimal number of multiplications to multiply the given chain is 72; by recording the split values, we see that the optimal order of multiplication is $(((A_1 \cdot A_2) \cdot A_3) \cdot A_4)$.

□

3. A hiker has a choice of n objects $\{o_1, o_2, \ldots, o_n\}$ to fill a knapsack of capacity W. Object o_i has benefit p_i and weight w_i. A subset of objects is said to be feasible if the combined weight of the objects in the subset is at most W. The hiker's goal is to select a feasible subset of objects such that the benefit to him is maximized (benefits are additive). Note that an object cannot be selected fractionally; it is either selected or not. Design a dynamic program to help the hiker.

   Solution: We first formulate the problem as an integer program. Let x_i denote a variable, which is 1 if the hiker selects object o_i and 0, otherwise.

   Accordingly, the benefit to the hiker is: $\sum_{i=1}^{n} p_i \cdot x_i$ and the cumulative weight of the objects in the knapsack is: $\sum_{i=1}^{n} w_i \cdot x_i$. Since, the capacity constraint of the knapsack cannot be violated, we must have $\sum_{i=1}^{n} w_i \cdot x_i \leq W$.

   Thus the integer program is:

$$\max \sum_{i=1}^{n} p_i \cdot x_i$$
$$\sum_{i=1}^{n} w_i \cdot x_i \leq W.$$

   Let S_i denote that subset of the set of objects, that includes the first i objects only. In other words, $S_0 = \emptyset$, $S_1 = \{o_1\}$, $S_5 = \{o_1, o_2, \ldots, o_5\}$ and so on.

   Define $m[i,w]$ to be the maximum benefit that can be reaped using only the objects in S_i and a knapsack of capacity w. The following recurrences follow naturally:

$$\begin{aligned} m[0,w] &= 0 \\ m[i,0] &= 0 \\ m[i,w] &= m[i-1,w], \text{ if } w_i > w \\ m[i,w] &= \max\{m[i-1,w], m[i-1,w-w_i] + p_i\}, \text{ if } w_i \leq w \end{aligned}$$

   The first equality states that if there are no objects at all to choose from, then regardless of the capacity of the knapsack, the benefit reaped is 0. Likewise, the second equality states that if the knapsack has capacity 0, it does not matter how many objects you select; the accrued benefit is 0. The third equality states that if the o_i has weight exceeding the capacity of the knapsack, then we might as well focus on the first $(i-1)$ objects. If $w_i \leq w$, then we have two choices, viz., we could exclude object o_i, in which case the benefit accrued is $m[i-1,w]$ or we could include object o_i, in which case the benefit accrued is $m[i-1,w-w_i] + p_i$.

As with the other dynamic programs that we studied, we can construct a table for $m[i, j]$ and compute $m[n, W]$, which is the entry that we are interested in. Since each entry can be computed in $O(1)$ time, and there are $(n+1) \cdot W$ entries in the table, we can implement the dynamic program in $O(n \cdot W)$ time.

□

4. Let T denote a binary search tree. Show that

   (a) If node a in T has two children, then its successor has no left child and its predecessor has no right child.

   (b) If the keys in T are distinct and x is a leaf node and y is x's parent, then $y \cdot key$ is either the smallest key in T larger than $x \cdot key$, or the largest key in T smaller than $x \cdot key$.

   Solution:

   (a) Since a has two children, its right subtree is non-empty. Its successor therefore is the leftmost node in the right subtree. (Why? For instance, why cannot its successor be one of its ancestors?) But by definition, the leftmost node has no left child.

   A similar argument shows that the predecessor of a has no right child.

   (b) Since the keys in T are distinct, either $x \cdot key < y \cdot key$, or $y \cdot key < x \cdot key$.

   Assume that $x \cdot key < y \cdot key$; it follows that x is the left child of y. Furthermore, assume the existence of a node z in T, such that $x \cdot key < z \cdot key < y \cdot key$. Let r denote the root of T. Clearly, z must be in the same subtree of r (left or right) as y. (Why?) Then observe, that z must be an ancestor of y, because it cannot be part of the left subtree of y (x is the only node in the left subtree), and at the same time, $z \cdot key < y \cdot key$, i.e., z cannot be part of the right subtree of y either. Finally note that y cannot be part of the left subtree of z, since $y \cdot key > z \cdot key$ and y cannot be part of the right subtree of z, since that would imply that $x \cdot key > y \cdot key$, contradicting our initial assumption. In other words, such a z cannot exist. We conclude that $y \cdot key$ is the smallest key in T, that is larger than $x \cdot key$.

   A similar argument applies to the case in which $y \cdot key < x \cdot key$.

□

5. An AVL tree is a binary search tree that is height balanced: for each node x, the heights of the left and right subtrees of x differ by at most 1. Prove that an AVL tree with n nodes has height $O(\log n)$.

   Solution: We need the following lemma.

   Lemma 1.1 *In an AVL tree of height h, there are at least F_h nodes, where F_h is the h^{th} Fibonacci number.*

   Proof: The base $h = 0$ is easy: a tree of height 0 has either 1 or 0 nodes, and $F_0 = 0$.

   Now assume that the claim is true for all AVL trees of height h. Let T be an AVL tree of height $h+1$. Let T_l and T_r be the left and right subtrees of the root of T, respectively. The AVL condition implies that T_l and T_r are both AVL trees. Since T has height $h+1$, at least one of T_l, T_r has height h and neither has a height greater than h. Thus, using the AVL condition, either both have height h, or one has height h and the other has height $h-1$. By induction, the number of nodes in T is at least

   $$F_h + F_{h-1} + 1,$$

   which is at least F_{h+1}. □

   Now we use a standard fact about the Fibonacci numbers:

   $$F_{i+2} \geq \phi^i \text{ for } i \geq 0$$

   where $\phi (\approx 1.61803399)$ is the *golden ratio* (see Section 4-5 of [CLRS09]).

   Combining Lemma 1.1 and this fact, we see that an AVL tree of height $h \geq 2$ has at least ϕ^{h-2} many nodes. This implies that an AVL tree with n nodes has height at most

   $$\left\lfloor \frac{\log n}{\log \phi} + 2 \right\rfloor,$$

which is $O(\log n)$.

□

References

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 3^{rd} edition, 2009.