Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
S. Daum, A. Ghodselahi, O. Saukh November 19, 2013

Algorithm Theory, Winter Term 2013/14
Problem Set 3

hand in by Thursday, November 28, 2013

Exercise 1: Sub-Sequences

(a) Consider a sequence $A = a_1, a_2, \ldots, a_m$ of characters. A sub-sequence S of A is a sequence that can be obtained by deleting some characters in A. Hence, for example, a, r, e, w, a, a is a sub-sequence of $b, a, a, r, e, z, w, a, b, a$. Given two sequences A of length m and B of length n, describe an $O(m \cdot n)$ time dynamic programming algorithm to compute the length of a longest common sub-sequence of A and B! To describe the algorithm, specify the sub-problems you need to solve (and store in a table), the initialization, as well as the recursive rule to compute the sub-problems.

(b) Apply your algorithm to the inputs $A = a, e, a, a, g, s, s, t$, and $B = b, a, e, b, s, a, g, t$ by setting up and filling out the table you need. Explain how you get the length of the longest common sub-sequence from your table. Also show how to compute a longest common sub-sequence of A and B.

(c) For two character sequences A and B, a super sequence C is a character sequence such that both A and B are sub-sequences of C. Give an efficient algorithm for finding the shortest common super-sequence of two strings A and B.

Hint: *Try to find a connection between the longest common sub-sequence of A and B and the shortest common super-sequence of A and B.*

Exercise 2: Sharing an Inheritance

Assume that Alice and Bob are two siblings who inherit some goods from their deceased parents. In order to share the inherited goods in a fair way, Alice and Bob decide about a value v_i for each inherited item i and they would like to share the goods so that both of them get items of the same total value.

Assume that there are items $1, \ldots, n$ and that each item i has an integer value $v_i > 0$. Give a dynamic programming algorithm that determines whether the n items can be partitioned into two parts of equal total value. If the items can be partitioned like this, the algorithm should also allow to compute such a partition. What is the time complexity of your algorithm? What assumptions do you need to get an algorithm that runs in time polynomial in the total number of items n?

Exercise 3: Stacking Boxes

We are given a set of n rectangular 3-D boxes, where the i^{th} box has length ℓ_i, width w_i, and height h_i, and where all three dimensions are *positive integers*. We further assume that for each box i, $\ell_i \geq w_i$. The objective is to create a stack of boxes which has *a total volume that is as large as possible*, but one is only allowed to stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Hence, it is only possible to put box j on top of box i if $\ell_i > \ell_j$ and $w_i > w_j$. See Figure 1 as an illustration.

(a) As a first attempt, we might try to use a greedy algorithm. Because when stacking boxes on top of each other, the lengths have to be strictly decreasing, a natural strategy might be to first sort the boxes by decreasing length such that $\ell_1 \geq \ell_2 \geq \ldots \geq \ell_n$. Then, we go through the boxes in that order and add a box on the existing stack whenever possible. Show that the behavior of this greedy algorithm can be arbitrarily bad!

(b) Instead of sorting the boxes by length, we could alternatively sort by decreasing area of the 2-D base. That is, we sort the boxes such that $\ell_1 w_1 \geq \ell_2 w_2 \geq \ldots \geq \ell_n w_n$. Show that now, box j can only be placed on top of box i if $j > i$ and therefore $\ell_j w_j \leq \ell_i w_i$. We again consider the greedy algorithm, but this time we go through the boxes according to the new ordering where the boxes are sorted by decreasing base area. We further simplify and assume that *all boxes have height* $h_i = 1$. Show that if there is a stack of volume V, the greedy algorithm finds a stack of volume at least $\Omega(V^{2/3})$.

(c) As a greedy algorithm does not seem to lead to a solution of the problem, let us finally try another approach. We again consider the general case where the heights of the boxes are arbitrary integers. Give an efficient algorithm that solves the box stacking problem by using dynamic programming. (The algorithm should return a maximum volume stack of boxes.) What is the running time of your algorithm? (as a function of n)

Hint: *It might still make sense to go through the boxes in order of decreasing base area.*

Remark: This was an exam question in Fall 2013.

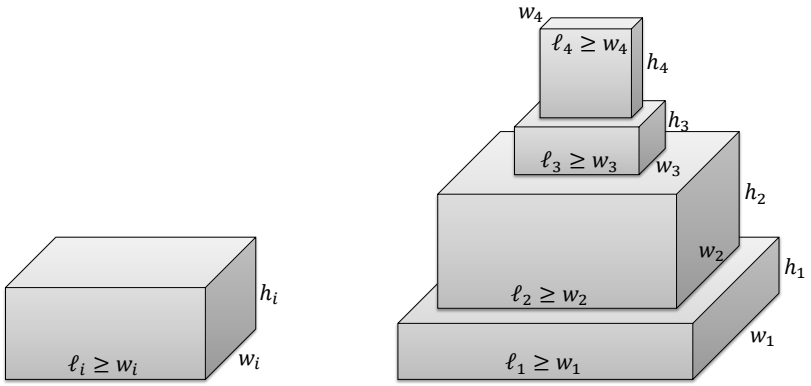

Figure 1: A single box of length ℓ_i, width w_i, and height h_i, as well as a feasible stack of boxes.