MCS 401 – Computer Algorithms I
Fall 2019
Problem Set 4

Lev Reyzin

Due: 10/28/19 by the beginning of class

1. Consider the following coin changing problem. You are given a value N and an infinite supply of coins with values $d_1, d_2, \ldots, d_k$. Give an $O(Nk)$ algorithm for finding the smallest number of coins that add up to the value N.

2. Call string C an *interleaving* of strings A and B if it contains all (and only) the characters of both A and of B and if their respective order is preserved in C. For example, $C = aacabbaa$ is an interleaving of $A = aaba$ and $B = caba$ (demonstrated as follows: **aa***cab***ba***a*). Give an algorithm that, given strings A, B, and C, decides whether C is an interleaving of A and B in polynomial time. Prove your answer correct.

3 A subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements; e.g. "acef" is a subsequence of "abcdef." Consider the problem of finding the longest common subsequence of two sequences – this is a task versioning systems like git or cvs often solve. Show that this is a special case of the sequence alignment problem. Then, give a polynomial-time algorithm for finding the longest subsequence common to *three* sequences. Analyze its running time and argue why it is correct.

4. You are given a one-dimensional array that may contain both positive and negative integers. The goal is to find the sum of contiguous (ie. next to one another, in sequence) subarray of numbers which has the largest sum. For example, if the given array is $[-2, -5, \mathbf{6}, \mathbf{-2}, \mathbf{-3}, \mathbf{1}, \mathbf{5}, -6]$, then the maximum subarray, marked in boldface, has sum equal to 7. On problem set 3, we saw that this problem had an $O(n \log n)$ divide-and-conquer solution, but faster algorithms exist. Give an $O(n)$ dynamic programming algorithm for this problem and argue for its correctness.

5. Give an algorithm that counts the number of distinct ways a number N can be written as the sum of the numbers 1, 3, and 5 that runs in time $O(N)$. Argue that your algorithm is correct.

(Note that N is the value of the number, not the input size.)

For example,
$$\begin{aligned} 6 &= 1+1+1+1+1+1 && (1)\\ &= 3+1+1+1 && (2)\\ &= 1+3+1+1 && (3)\\ &= 1+1+3+1 && (4)\\ &= 1+1+1+3 && (5)\\ &= 3+3 && (6)\\ &= 5+1 && (7)\\ &= 1+5 && (8) \end{aligned}$$

So, for $N = 6$ the answer is 8.

6. You are given a complete binary tree on $n = 2^d - 1$ vertices (for $d \geq 1$), rooted at vertex r. Further, each vertex i in the tree is assigned a weight $w_i > 0$. The problem is to find the k-vertex subtree[1] (with $1 \leq k \leq n$) rooted at r for which the sum of the weights of its included vertices is maximized. Give an algorithm that does this in time polynomial in n and k and argue for its correctness.

[1] A subgraph of a graph G is another graph formed from a subset of the vertices and edges of G. The vertex subset must include all endpoints of the edge subset, but may also include additional vertices. A subtree is a connected subgraph of a tree.