Problem Set 13

Data Structures and Algorithms, Fall 2021

You do NOT need to submit solutions for this problem set.

Problem 1

(a) Imagine that you wish to exchange one currency for another. You realize that instead of directly exchanging one currency for another, you might be better off making a series of trades through other currencies, winding up with the currency you want. Suppose that you can trade n different currencies, numbered $1 \cdots n$, where you start with currency 1 and wish to wind up with currency n. You are given, for each pair of currencies i and j, an exchange rate r_{ij}, meaning that if you start with d units of currency i, you can trade for $d \cdot r_{ij}$ units of currency j. A sequence of trades may entail a commission, which depends on the number of trades you make. Let c_k be the commission that you are charged when you make k trades. Show that if commissions c_k are arbitrary values, then the problem of finding the best sequence of exchanges from currency 1 to currency n does not necessarily exhibit optimal substructure.

(b) Give a dynamic-programming algorithm to the 0-1 knapsack problem that runs in $O(nW)$ time, where n is the number of items and W is the maximum weight of items that the thief can put in his knapsack. Is this algorithm a polynomial-time algorithm? Justify your answer.

Problem 2

This problem asks you to devise algorithms to compute optimal subtrees in *unrooted* trees—connected acyclic undirected graphs. In this problem. a *subtree* of an unrooted tree is any connected subgraph.

(a) Suppose you are given an unrooted tree T with weights on its *edges*, which may be positive, negative, or zero. Describe and analyze an algorithm to find a *path* in T with maximum total weight. Your algorithm only need to return the weight value. You may assume a path only contains distinct vertices. For full credit, your algorithm should have $O(n)$ runtime assuming T contains n vertices.

(b) Suppose you are given an unrooted tree T with weights on its *vertices*, which may be positive, negative, or zero. Describe and analyze an algorithm to find a subtree of T with maximum total weight. Your algorithm only need to return the weight value. For full credit, your algorithm should have $O(n)$ runtime assuming T contains n vertices. (This was a 2016 Google interview question.)

(We only ask for returning the weight value so as to keep the code cleaner. Nonetheless, you should also think about how to efficiently reconstruct the optimal solution. It's another good exercise!)

Problem 3

Describe and analyze an efficient algorithm to find the length of the longest *contiguous* substring that appears both forward and backward in an input string $T[1 \cdots n]$. The forward and backward substrings must *not* overlap. Here are several examples:

- Given the input string `ALGORITHM`, your algorithm should return 0.
- Given the input string `RECURSION`, your algorithm should return 1, for the substring `R`.

- Given the input string `REDIVIDE`, your algorithm should return 3, for the substring `EDI`. (The forward and backward substrings must not overlap!)
- Given the input string `DYNAMICPROGRAMMINGMANYTIMES`, your algorithm should return 4, for the substring `YNAM`. (In particular, it should not return 6, for the subsequence `YNAMIR`).

Problem 4

Suppose you are given a sequence of integers separated by $+$ and $-$ signs; for example:

$$1 + 3 - 2 - 5 + 1 - 6 + 7$$

You can change the value of this expression by adding parentheses in different places. For example:

$$\begin{aligned} 1 + 3 - 2 - 5 + 1 - 6 + 7 &= \quad -1 \\ (1 + 3 - (2 - 5)) + (1 - 6) + 7 &= \quad 9 \\ (1 + (3 - 2)) - (5 + 1) - (6 + 7) &= \quad -17 \end{aligned}$$

Describe and analyze an algorithm to compute, given a list of integers separated by $+$ and $-$ signs, the maximum possible value the expression can take by adding parentheses. Parentheses must be used only to group additions and subtractions; in particular, do not use them to create implicit multiplication as in $1 + 3(-2)(-5) + 1 - 6 + 7 = 33$.