INF 4130 Exercises, Sept. 4 and 9, 2014

Exercise 1

1.1 Solve exercise 20.19 (B&P); the solution to (a) was shown in the lecture, but solve (b).

1.2 Run the algorithm (on paper) with two similar words, e.g., "algori" og "logari", and with two identical words.

1.3 Here you should go through the argument for why using the formula in Chapter 20.5 (and many places in the slides) will solve the *edit distance problem* correctly. This amounts to showing that the DP requirement holds when using this formula as the *Combine* function. Why this is true is not very well explained, neither in the book nor in the slides used at the lecture. We will have to look separately at case 1 and case 2 (see slides), and show that:

Case 1 (P[i] =T[j]): If D[i-1, j-1] is optimal (for the corresponding smaller problem), then the computed value for D[i, j] is optimal for that problem.

Case 2 (P[i] != T[j]): If D[i-1, j-1,], D[i, j-1], and D[i-1, j] are all optimal (for the corresponding smaller problems), then the computed value for D[i, j] is optimal for that problem.

Hint (also given at slide 4): Assume that D[I, j], computed by the formula, will *not* be optimal. Then show that you will end up in a contradiction.

1.4 Show how to implement the algorithm using only one column (or row) plus a few additional variables.

Exercise 2

Look into memoization – using a table as in standard dynamic programming, but with an algorithm following the recursive formula top-down. The trick is now that each recursive call first looks into the table, and checks if the answer to the current sub-problem is already calculated. If it is, this value is used, otherwise we have to do recursive calls to solve the necessary smaller problems.

Write such an algorithm for exercise 20.19.

Exercise 3 (if there is time)

We are going to solve the Money-Changing Problem using dynamic programming: The problem is to return K pence (cents/øre, or whatever) using as few coins as possible. Let $\{v_1, v_2, \ldots, v_n\}$, be the coin denominations, for instance, $v_1 = 25$, $v_2 = 10$, $v_3 = 5$, $v_4 = 1$. We assume the denominations are integer, and that $v_{i-1} < v_i$. We further assume $v_n = 1$, so that a solution always exists. We do, of course, always have enough coins.

For the currency described above, a greedy solution will work. (And it is an interesting exercise to prove just that.) BUT if we remove the 5 pence coin, we end up in trouble: A greedy algorithm returning change for 30 pence, will return one 25 pence coin, and five pennies, six coins in total, twice as many as the optimal three 10 pence coins. Dynamic programming finds optimal solutions for all possible currencies, including those where greedy algorithms fail.

Let $C[j]$ be the number of coins in a solution where we return change for j pence. If a solution uses a coin of denomination v_i, we have $C[j] = C[j - v_i] + 1$.

a) Give a recursive formula for the value of an optimal solution.

b) Write (pseudo) code for an algorithm based on the formula from a).

c) Run the algorithm with $v_1 = 30$, $v_2 = 24$, $v_3 = 12$, $v_4 = 6$, $v_5 = 3$, $v_6 = 1$, og $K = 48$. (English coins, not too long ago!) [The greedy solution is 30+12+6, by the way.]

d) What is the running time of your algorithm?