IN3130
Dynamic Programming problems

Exercise 1

Look at the problem of finding the "best path" (least weight) from upper left to lower right corner in some of the first slides. Make sure that everybody understands what is going on, by discussing the following points:

a) Indicate in which different orders matrix P can be filled out, if we want to have the necessary results ready when we need them?
b) How can we find the shortest path itself?
c) What is the complexity of this algorithm?

Exercise 2

a) Run the "Edit Distance Algorithm" (on paper) with two similar words, e.g., "algori" og "logari", and with two identical words.
b) Show how to implement the algorithm using only one column (or row) plus a few additional variables.
c) Solve the problem given in the last sentence of section 20.5 on page 645. That is: In the slides we originally wanted to find an algorithm for searching through a string T, and look for substrings S = T[p], T[p+1], … , T[q] of T similar to a given string P. We can assume that we want to find the first substring of T whose edit distance to P is less than or equal to a given K (or report that no such substring occurs).

Exercise 3

a) 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 finding the edit distance between two strings P and T.

Exercise 4

The Fibonacci numbers F(n) are defined by the formulas

F(0)=0, F(1)=1 and F(n) = F(n-1) + F(n-2) for n > 1

One can compute F(n) for a given n by building up the sequence of F(0), F(1), F(2), F(3), …, F(n) like this:

0 1 1 2 3 5 8 13 21 34 55 89 …

a) This computation can be seen as DP-computation where we have a one dimensional table holding all the values we have already computed, and use some of those to compute the next. Of which order is this computation when expressed in O-notation of the value n.

b) In what sense is it *not* reasonable to say that this is a polynomial algorithm. Compare with the speed of adding by hand two numbers n and m. What if this computation used time e.g. $O(m + n)$?
COMMENT: This sort of "polynomial time algorithm" is often said to run in "quasi-polynomial time".

c) Assume we used the formula F(n)= F(n-1) + F(n-2) to write a simple recursive program for F(n), without trying to remember any previously computed values. What would be the execution time for such a program?