IN3130 Exercise set 4

Dynamic Programming Problems

Exercise 1

Look at the problem of finding the "best path" (lowest weight) from the upper left to the lower right corner. (First example from the lecture.) Make sure everybody understands what is going on, by discussing the following points:

a) Indicate what different orders the matrix P can be filled out, if we want to have the necessary results when we need them.
b) How can we find the shortest path itself, not only its weight?
c) What is the (time) complexity of this algorithm?

Exercise 2

a) Run the Edit Distance algorithm (on paper) with two similar words, e.g. "algori" and "logari", and with two identical words.
b) Show how to implement the Edit Distance algorithm using only one row or one column plus a few extra variables.
c) On the lecture slides (the example on slide 12) we looked at the output from an algorithm for searching through a string T, looking for substrings S = T[p], T[p+1], …, T[q] of T that are similar to a string P. The example says nothing about how similar is similar enough. It was just used as an intro to the Edit Distance concept. Describe an algorithm that finds the first substring of T whose Edit Distance to P is less than or equal to a given K (or report that no such string occurs).

Exercise 3

Look into Memoization – using an algorithm following a recursive formula top-down, while storing solutions to sub-problems in a table like in standard Dynamic Programming. The memorization trick is that each recursive call first checks the table, to see if the solution to a sub-problem has been calculated already. If it is, that value is used. Otherwise, we have to do recursive calls to solve the necessary (smaller) subproblems.

a) Write a memorized algorithm for finding the edit distance between sting P and T.

Exercise 4

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

F(0) = 0
F(1) = 1
F(n) = F(n-1) + F(n-2), for $n \geq 2$

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

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

a) This computation can be seen as Dynamic Programming computation where we have a one-dimensional table holding all the values we have already computed, and use some of them to compute the next value. 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 that 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?

[end]