Reading material

At the lecture we will continue with dynamic programming. We will talk about the knapsack problem and sequence alignment. You should read KT section 6.4 and 6.6.

Exercises

$[w]$ means it is a warmup exercise and $[*]$ that it is an extra difficult exercise.

1 $[w]$ **Knapsack** Solve the following knapsack problem by filling out the table below. The items are givens as pairs (w_i, v_i): $(5,7),(2,6),(3,3),(2,1)$. The capacity $W = 6$.

4								
3								
2								
1								
0								
$i \setminus w$	0	1	2	3	4	5	5	6

2 Sequence alignment $[w]$ Consider the strings `APPLE` and `PAPE` over the alphabet $\Sigma = \{\texttt{A},\texttt{E},\texttt{L},\texttt{P}\}$ and a penalty matrix P:

	A	E	L	P
A	0	1	3	1
E	1	0	2	1
L	3	2	0	2
P	1	1	2	0

Compute the sequence alignment of the two strings when the penalty for a gap $\delta = 2$. Fill the dynamic programming table below, and explain how the minimum cost sequence alignment is found in it.

	j	0	1	2	3	4	5
i			A	P	P	L	E
0							
1	P						
2	A						
3	P						
4	E						

3 Longest palindrome subsequence A *palindrome* is a (nonempty) string over an alphabet Σ that reads the same forward and backward. For example are `abba` and `racecar` palindromes. A string P is a *subsequence* of string T if we can obtain P from T by removing 0 or more characters in T. For instance, `abba` is a subsequence of `bcadfbbba`.

Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. To do this first give a recurrence for the problem and then write pseudo code for an algorithm based on your recurrence and dynamic programming. Argue that your recurrence is correct and analyse the running time and space usage of your algorithm.

4 Kattis Exercises Solve the following problems on Kattis:

- Knapsack: https://open.kattis.com/problems/knapsack
- Baas: https://open.kattis.com/problems/baas

  Before you start programming, first design an algorithm that solves the problem and analyse its running time.

5 Defending Zion Solve KT 6.8

6 Gerrymandering $[*]$ Read KT 6.24.
Consider the following function $G[j,k,d_1,d_2] = \mathit{true}$ if it is possible to allocate k of the first j precincts to district 1 and at the same time achieve at least d_1 votes for party A in district 1 and at least d_2 votes for party A in district 2. Otherwise $G[j,k,d_1,d_2]$=*false*.

6.1 Explain how $G[j,k,d_1,d_2]$ can be used to find the solution (assuming you have already computed the solution for all relevant parameters ($j = 1,\ldots,n$, $k = 1,\ldots,?$, etc.)).

6.2 Write a recursion for $G[j,k,d_1,d_2]$.

6.3 Write pseudo code implementing a dynamic programming algorithm that solves the problem based on your recursion.

6.4 Analyse the space and running time of your algorithm.

Puzzle of the week: 100 Ants [1]There are 100 ants on a rod of length 1 metre. The ants are arbitrarily positioned on the rod and are travelling at 1 metre/minute either right or left at the start. The ants are also perfectly elastic, so that if two ants collide they simply turn round and carry on at 1 metre/minute in the opposite direction. If an ant reaches the end of the rod it falls off. The question is, what is the longest time it can take for all 100 ants to fall off the rod?

You can assume the ants are points on the rod and that the rod is simply a one dimensional line (i.e. ants can only go left or right). You are asked to find the longest time it can take for all the ants to fall off the rod over all possible start states and starting directions of the ants.

[1]I got this puzzle from Raphäel Clifford