



INF2220: algorithms and data structures

Series 13

Topic Dynamic programming

Issued: 16. 11. 2017

Classroom

Exercise 1 (Fibonacci) Design an algorithm to compute Fibonacci numbers in *linear* time.

Exercise 2 (Matrix) Let A be an N -by- N matrix of zeros and ones. A submatrix S of A is any group of contiguous entries that forms a square. Design an algorithm that determines the number of elements of the largest submatrix of “ones” in A . For instance, in the matrix that follows, the largest submatrix is a 4-by-4 square.

```
10111000
00010100
00111000
00111010
00111111
01011110
01011110
00011110
```

Exercise 3 Product-sum (from 2014 exam)

Given a list of n integers, v_1, \dots, v_n , the *product-sum* of the list is the *largest* sum that can be formed by multiplying adjacent elements in the list. Each element can be matched with at most one of its neighbors. For example, the product-sum of the list 1, 2, 3, 1 is 8 ($= 1 + (2 \times 3) + 1$) and the product-sum of the list 2, 2, 1, 3, 2, 1, 2, 2, 1, 2 is 19 ($= (2 \times 2) + 1 + (3 \times 2) + 1 + (2 \times 2) + 1 + 2$).

1. Optimization

Given a list of n integers for $n \geq 2$. What is the optimal product-sum $OPT(j)$ for the first j elements in the list? Meaning, you should find out what $OPT(j)$ is for $j \in \{0, 1, \dots, n\}$.

2. Dynamic programming

Give a dynamic programming solution for this problem. Implement the method `prodSum` which returns the product-sum of the first j elements of a list v :

```
int prodSum(int [] v, j) {
    .....    // your code
}
```

The method only needs to return the product-sum.

Exercise 4 Assume that you have a table with size $n \times m$, with positive values in each cell. Further, assume that you start in the left upper corner, and that you are allowed to go only to the right or downwards. For each cell you enter, you add the cell's value to your counter. Use dynamic programming to find the maximum value you can get by taking any route from your position to the lower right corner.

Exercise 5 (Skip this if you not are familiar with matrix multiplication)

Assume that you are going to multiply n matrices in a given sequence. This can be done in many ways; e.g. as $(AB)C$ or $A(BC)$. The sequence you use will determine the number of multiplications and additions (floating point operations) you need to perform the multiplication. Use dynamic programming to find the most optimal sequence. (You can assume that all multiplications are well defined, i.e. that the dimensions matches. The number of multiplications needed to multiply two matrices of size $(l \times m), (m \times n)$ is of order $O(lmn)$.)

Lab

Exercise 6 Create a simplified implementation of a *Huffman* compression where you:

1. Reads the input file and create the frequency table for each letter.
2. Use the frequency table plus a binary heap to create Huffman tree.
3. Saves the bit representation of characters as String (ex: "001").
4. Print `{letter, frequency, binary representation}` and about how much space would be saved by compressing the file with your Huffman tree.

Exercise 7 The *longest common subsequence* problem is as follows: Given two sequences $A = a_1, a_2, \dots, a_m$, and $B = b_1, b_2, \dots, b_n$, find the length, k , of the longest sequence $C = c_1, c_2, \dots, c_k$ such that C is a subsequence (not necessarily contiguous) of both A and B . As an example, if

$$\begin{aligned} A &= \text{d,y,n,a,m,i,c} \\ B &= \text{p,r,o,g,r,a,m,m,i,n,g,} \end{aligned}$$

then the *longest common subsequence* is `a,m,i` and has a length 3. Give an algorithm to solve the *longest common subsequence* problem. Your algorithm should run in $O(MN)$ time.

References