CSE100 Algorithm Design and Analysis — Homework set #3
Fall semester 2018 — Miguel Á. Carreira-Perpiñán

Total possible marks: 100. You have to submit this homework to a TA due Nov. 21 at 8 pm or at your own lab session (also at TA's office hours if it is before the deadline). Homeworks must be done in groups of up to 3 students (one submission per group, listing materials consulted). This set covers chapters 15–16 of the textbook *Introduction to Algorithms*, 3rd. ed., by Cormen et al.

Note: for all numerical exercises, give only the final result. However, give a brief explanation of how you do the computation. For example, for exercise 1.2 about the matrix-chain multiplication, the caption of fig. 15.5 in the textbook illustrates how entry $m[2, 5]$ is computed.

Exercise 1: matrix-chain multiplication (25 points).

1. (7 points) (Exercise 15.2-2 in the textbook.) Give a recursive algorithm MATRIX-CHAIN-MULTIPLY(A, s, i, j) that actually performs the optimal matrix-chain multiplication, given the sequence of matrices $\langle A_1, A_2, \ldots, A_n\rangle$, the s table computed by procedure MATRIX-CHAIN-ORDER, and the indices i and j. (The initial call would be MATRIX-CHAIN-MULTIPLY$(A, s, 1, n)$.) Assume you can use a function MATRIX-MULTIPLY(A, B) that returns a matrix $C = AB$.

2. (10 points) Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is $\langle 6, 10, 3, 50, 5\rangle$. Fill in the m and s tables as in fig. 15.5 in the textbook.

3. (8 points) Give the result obtained using a greedy algorithm that picks the smallest local cost $p_{i-1}p_kp_j$ over $i \leq k < j$. Comment on whether this greedy algorithm always solves correctly the matrix-chain multiplication problem or not.

Exercise 2: 0–1 knapsack problem (50 points). The 0–1 knapsack problem consists of, given a knapsack that can hold a maximum weight of W and n items $i = 1, \ldots, n$ with weights w_i and values v_i, finding a subset of items with total weight less or equal than W and maximum total value. The weights $W, w_1, \ldots, w_n$ are positive integers and the values $v_1, \ldots, v_n$ are positive but not necessarily integer.

1. (3 points) Give a lower bound on the run time of the brute-force approach that examines all possible solutions.

Solve the 0–1 knapsack problem using dynamic programming:

2. (10 points) Write the value of an optimal solution as a recursive expression using the value of optimally solved subproblems. *Hint*: define a subproblem given by a total weight and a subset containing the first j items, and consider the choice of having or not having item j in an optimal solution for that subproblem.

3. (8 points) Write pseudocode that implements this solution top-down using memoization, and prints the solution. Give its run time.

4. (8 points) Write pseudocode that implements this solution bottom-up by filling in a table in order, and prints the solution. Give its run time.

Consider a knapsack example with total weight $W = 5$ and possible items

item i	1	2	3	4	5	6
value v_i	\$2	\$3	\$7	\$5	\$11	\$6
weight w_i	2	2	4	3	6	5

5. (7 points) Give the tables that the bottom-up algorithm would construct and so give an optimal solution.

6. (7 points) Give the recursion tree of the calls in the top-down algorithm using memoization.

7. (3 points) In general for the 0–1 knapsack problem, which version is faster, top-down using memoization or bottom-up by filling in the table? *Hint*: consider which subproblems are computed.

8. (4 points) Give the solution assuming the following greedy algorithm: we repeatedly take the item (all of it) with most value per unit of weight, until either the knapsack is full or there are no items left. Comment on the result.

Exercise 3: Huffman codes (25 points). The following table gives the frequencies of each character in a file with 1 000 characters.

α	β	γ	δ	ϵ	ζ	η	θ
100	20	300	50	400	80	10	40

1. (2 points) How many bits do we need to store the file if using a fixed-length code?

2. (8 points) Show the code built by the Huffman algorithm, both as a tree and as a list (character, codeword).

3. (2 points) How many bits do we need to store the file with this Huffman code?

Assume now that all the the characters have the same frequency (= 125).

4. (8 points) Show the code built by the Huffman algorithm, both as a tree and as a list (character,codeword).

5. (2 points) How many bits do we need to store the file with this Huffman code?

6. (3 points) Generalizing from this example, what can you say about the code built by the Huffman algorithm for an alphabet with $n = 2^b$ characters each of which occurs with equal frequency f? Explain your answer.

Bonus exercise: longest common subsequence (15 points). Construct the matrices c and b as in fig. 15.8 in the textbook for the words SPANKING and AMPUTATION, and give a longest common subsequence for them.