CS170 Fall 2013 Solutions to Homework 8

Zackery Field, section Di, 103, `cs170-fe`

October 31, 2013

1. 6.4

You are given a string of n characters $s[1, \ldots, n]$, which you believe to be a corrupted text document in which all punctuation has vanished. You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function $dict()$: for any string w,

$$dict(w) = \begin{cases} true & \text{if } w \text{ is a valid word} \\ false & \text{otherwise} \end{cases}$$

(a) Give a dynamic programming algorithm that determines whether the string s can be reconstructed as a sequence of valid words. The running time should be at most $O(n^2)$, assuming that the calls to dict take unit time.

(b) In the event that the string is valid, make your algorithm output the corresponding sequence of words.

Continued on Page 6

2. 6.14 Cutting Cloth

You are given a rectangular piece of cloth with dimensions $X \times Y$, where X and Y are positive integers, and a list of n products that can be made using the cloth. For each product $i \in [1, n]$ you know that a rectangle of cloth of dimensions $a_i \times b_i$ is needed and that the final selling price of the proucdt is c_i. Assume that a_i, b_i, and c_i are all positive integers. You have a machine that can cut any rectangular piece of cloth into two pieces either horizontally or vertically . Design an efficient dynamic programming solution that determines the best strategy for cutting the $X \times Y$ piece of cloth, that is, a strategy for cutting the cloth so that the products made fomr the resulting pieces give the maximum sum of selling prices. You are free to make as many copies of a given product as you wish, or none if desired.

Continued on Page 7

3. 6.20 Optimal binary search trees

Suppose we know the frequency with which keywords occur in programs of a certain language. We want to organize them in a binary search tree, so that the keyword in the root is alphabetically bigger than all the keywords in the left subtree and smaller than all the keywords in the right subtree.

Figure 6.12 has a nicely-balanced example on the left. In this case, when a keyword is being looked up, the number of comparisons needed is at most three: for instance finding while, only the three nodes 'end', 'then', and 'while' get examined. But since we know the frequency with which keywords are accessed, we can use an even more fine-tuned cost function, the average number of comparisons to look up a word:

$$cost = lvl_1(w_{1_1} + \ldots + w_{1_n}) + \cdots + lvl_n(w_{n_1} + \cdots + w_{n_n})$$

By this measure, the best search tree is the one on the right, which has a cost of 2.18.
Give an efficient algorithm for the following task:
Input: n words (in sorted order); frequencies of these words: $p_1, p_2, \ldots, p_n$.
Output: The binary search tree of lowest cost.

Continued on Page 8

4. 6.29 Exon chaining

Each gene corresponds to a subregion of an overall genome (the DNA sequence); however , part of this region might be 'junk DNA'. Frequently, gene consists of several pieces called exons, seperated by junk fragments called introns. This complicates the process of identifying genes in a newly sequenced genome.

Suppose we have a new DNA sequence and we want to check whether a certain gene (a string) is present in it. Because we cannot hope that the gene will be a contiguous subsequence, we look for partial matches-fragments of DNA that are also present in the gene (actually, even these partial matches will be approximate, not perfect). We then attempt to assemble these fragments.

Let $x[1,\ldots,n]$ denote the DNA sequence. Each partial match can be represented by a triple (l_i, r_i, w_i), where $x[l_i,\ldots,r_i]$ is the fragment and w_i is the weight representing the strength of the match (it might be a local alignment score or some other statistical quantity). Many of these potential matches could be false, so the goal is to find a subset of the triples that are consistent (nonoverlapping) and have a maximum total weight.

Show how to do this efficiently.

Continued on Page 9

5. Timesheet Part 2

Recall problem 4 from homework 7. Suppose we have N jobs labelled $1, ..., N$. For each job, you have determined the bonus of completing the job, $V_i 0$, a penalty per day that you accumulate for not doing the job, $P_i 0$, and the days required for you to successfully complete the job $R_i > 0$.

Every day, we choose one unfinished job to work on. A job i has been finished if we have spent R_i days working on it. This doesnâĂŹt necessarily mean you have to spend R_i contiguous sequence of days working on job i. We start on day 1, and we want to complete all our jobs and finish with maximum reward. If we finish job i at the end of day t, we will get reward $V_i t$. Note, this value can be negative if you choose to delay a job for too long.

Now, what we did not tell you last time is that we have a time limit of T days, in which we can choose to work on some of these jobs in only those T days. Given this information, what is the optimal job scheduling policy with a time limit of T days? Notice that 0 is a lower bound since we can choose to do no jobs at all if all of them happen to have negative value, or all of them take more than time T.

Design an efficient dynamic programming algorithm that finds the optimal schedule.

Continued on Page 10

Extra space for Problem 1

Continued from Page 1

Extra space for Problem 2

Continued from Page 2

Extra space for Problem 3

Continued from Page 3

Extra space for Problem 4

Continued from Page 4

Extra space for Problem 5

Continued from Page 5