Algorithms Lab 7

(Dynamic programming)

Laura Toma, csci2200, Bowdoin College

This week's topics

- Dynamic programming: rod cutting and knapsack

Review the topics discussed in class this week. This is a great time to understand all the details and ask questions.

In-class (COLLABORATION LEVEL 0[1])

1. Argue that the following greedy strategies do not work for the 0-1 knapsack problem by giving a counter-example in each case:

   a. Select the items in order of their values; that is, select with largest value first, and so on.

   b. Select the items in order of their value-per-pound; that is, select the item with the largest value-per-pound first, and so on.

2. **A different variant of knapsack:** Same setup as the knapsack problem, except that there is an infinite supply of each item, and you can choose to take as many copies of each item as you want. Find the maximal value that you can put in the backpack.

Homework problems (COLLABORATION LEVEL 1[2])

1. Suppose you are in charge of planning a party for an institution (for e.g., Bowdoin College). The institution has a hierarchical structure, which forms a tree rooted at the President. On the very last level are the faculty, grouped by department. (I have such a chart in my office, if you are curious). Each faculty has below all students taking a class with him/her that particular semester. Assume that every person is listed at the highest possible position in the tree and there are no double affiliations: everybody has one and only one supervisor in this hierarchy and no student is in more than one class (this does not make sense, but for an institution it's mostly true).

[1]Collaboration level 0: everything allowed!

[2]Collaboration level 1: verbal collaboration without solution sharing. You are allowed and encouraged to discuss ideas with other class members, but the communication should be verbal and additionally it can include diagrams on board. Noone is allowed to take notes during the discussion (being able to recreate the solution later from memory is proof that you actually understood it). Communication cannot include sharing pseudocode for the problem. Check complete guidelines at: https://turing.bowdoin.edu/dept/collab.php)

You have access to a secret database which ranks each faculty/staff/student with a conviviality rating (a real number, which can be negative if the person is really grumpy or boring). As the party organizer, your goal is to maximize the fun, so you decide to select the guest list such that for any guest, his or her immediate supervisor is not also a guest. Note that this means that the President may not be invited, but it is what it is. You're doing your job.

You are given a tree that describes the strucure of the institution. Each node has a (down) pointer to its left-most child, and a (right) pointer to its next sibling (if unclear read Section 10.4 in CLR on tree representation). Each node also holds a name and a conviviality ranking. Describe an algorithm to make up a guest list that maximizes the sum of the conviviality rankings of the guests. Analyze the running time of your algorithm. Assume the size of the tree (number of nodes) is n.

Hint: Aim for $O(n)$ time.

2. Consider a quiz with n questions. For each $i = 1, 2, ..., n$, question i has integral point value $v_i > 0$ and requires $m_i > 0$ minutes to solve. Suppose further that no partial credit is awarded (unlike this exam).

   Your goal is to come up with an algorithm which, given $v_1, v_2, ..., v_n, m_1, m_2, ..., m_n$ and V, computes the minimum number of minutes required to earn at least V points on the quiz. For example, you might use this algorithm to determine how quickly you can get an A on the quiz.

   Let $M(i, v)$ denote the minimum number of minutes needed to earn v points when you are restricted to selecting from questions 1 through i.

   (a) Give a recurrence expression for $M(i, v)$. To get you started, here is the base case:
   - $M(i, v) = 0$ for all i, and $v \leq 0$.
   - $M(0, v) = \infty$ for $v > 0$

   (b) Write the recurrence for the worst-case running time of a straightforward implementation of (a) (without a table), and use it to get an $\Omega()$ expression for the worst case running time.

   (c) Give pseudocode for a dynamic programming algorithm to solve the problem.

   (d) Analyze its running time.

3. A pharmacist has W pills and n empty bottles. Bottle i can hold p_i pills and has an associated cost c_i. Given W, $\{p_1, p_2, ..., p_n\}$ and $\{c_1, c_2, ..., c_n\}$, you want to store all pills using a set of bottles in such a way that the total cost of the bottles is minimized.

   (a) Explain how the problem has optimal substructure, and how your algorithm will use it.

   (b) Give a recursive formulation of this problem (either as a function, or as pseudocode); if your function has arguments, make sure to explain what the arguments represent, and what your function will return. You do not need to prove correctness.

   (c) Describe a dynamic programming algorithm (eitehr recursive or iterative) and analyze its running time.

4. You have been hired to design algorithms for optimizing system performance. Your input is an array $J[1..n]$ where $J[i]$ or J_i represents the running time of job i; jobs do not have specific start and end times, but they can be started at any time (this is a different scenario than in interval scheduling). The running times are integers.

   Generally speaking, your task is to find an optimal load balance of these tasks over two processors.

   Design an algorithm for determining whether there is a subset S in J such that the running time of the elements in S sum up precisely to the same amount as the sum of the elements not in S; more formally, $\sum_{J_i \in S} J_i = \sum_{J_i \in J-S} J_i$. The algorithm should run in time $O(n \cdot N)$, where N is the sum of the running times of the n jobs.