Problem Set 3

Labs: Mar 7-9, 2012
Due: Mar 15, 2012 10:30am

All problem sets must be handed in **on paper** in class on the due date.

Please ensure that your solutions are complete, clear, and concise. Points will be deducted for overly complex solutions.

Instructions

For all of the problems we ask you to do the following:

- Solve some examples or prove some lemmas to give you an intuition for the problem.
- Design a dynamic programming algorithm. The first step in designing a dynamic programming algorithm is to isolate the subproblem in the form of a recurrence relation. Do not worry about efficiency for the recurrence relation. Focus on correctness and isolating the subproblem. For example for the Fibonacci sequence the correct recurrence relation would look like:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

  After identifying the recurrence relation your algorithm must also include details defining a dynamic programming array by specifying what set of parameters index each array entry and the meaning of each array entry. The last step is to define the order in which you will fill in the entries of the dynamic programming array.
- *Briefly* prove the correctness of your recurrence relation.
- *Briefly* analyze running time of your algorithm. Note that the running time of a dynamic programming algorithm is often the storage requirements times the complexity of one recursive step.

Problem 1

Your job is to assist in parking cars. You are given a sequence of n cars $[c_1, c_2, \dots, c_n]$ along with a compatibility matrix $P_{i,j}$ for $1 \le i \le j \le n$, where $P_{i,j}$ is *true* iff cars c_i and c_j can be parked next to each other and *false* otherwise. Your task is to determine if there exists a sequence of cars such that all of the compatibility requirements are satisfied, and with the additional locality requirement that each car can move at most 2 parking spots from where it is initially located (otherwise the car owners won't be able to find their vehicles). That is, c_r can be moved to the r-th, $(r \pm 1)$-th, or $(r \pm 2)$-th spot in the final sequence of parked cars.

a. Consider as input a sequence of cars $[c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}]$ where every car is compatible with every other car. If you have decided to place c_7 in parking spot number 5 then what cars can be parked before slot 5 and what cars can be parked after slot 5?

b. Let a *window* be a set of 4 consecutive parking spots $i-3, i-2, i-1, i$ where $4 \le i \le n$. Assume you know which cars are parked in the window (i.e. slots $i-3, i-2, i-1, i$). Show that, for all other cars you know whether they are parked before slot $i-3$ or after slot i.

c. Design a dynamic programming algorithm for the Car Parking problem which takes as input the initial sequence of cars $[c_1, ..., c_n]$ and the compatibility matrix P, and outputs *true* if a feasible sequence of cars exists, and otherwise outputs *false*, in $O(n)$ time.

d. Prove the correctness of your algorithm. As an indication, we expect that the proof will typically be about $\frac{3}{4}$ of a page long.

e. Prove that the running time of your algorithm is $O(n)$. As an indication, we expect that the proof will typically be only a few lines long.

Problem 2

You are a software engineer developing in an embedded system. The system manufacture has announced that there are a number of bugs in their architecture and only a subset of arithmetic expressions will be evaluated correctly. The manufacturer has provided the following recursive specification for acceptable arithmetic expressions (AAE),

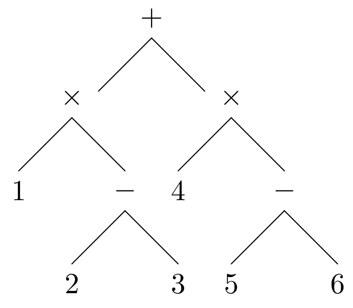
$$\begin{aligned} R_1 &: S_0 \to A \\ R_2 &: A \to B + B \\ R_3 &: B \to A \times B \\ R_4 &: B \to A - A \\ R_5 &: A \to \mathbf{N} \end{aligned}$$

Given an arithmetic expression w, your task is to determine whether w will be correctly evaluated by the embedded system. In other words, does there exists a number of applications of the recursive rules in AAE, starting with S_0, which yield w? (If you know computational theory, this problem is the same as word recognition in context-free grammars).

For example, the expression $1 \times 2 - 3 + 4 \times 5 - 6$ can be generated with the following derivation of AAE,

$\underline{S_0} \xrightarrow{R_1} \underline{A} \xrightarrow{R_2} \underline{B} + B \xrightarrow{R_3} A \times \underline{B} + B \xrightarrow{R_4} A \times A - A + \underline{B} \xrightarrow{R_3} A \times A - A + A \times \underline{B} \xrightarrow{R_4} \underline{A} \times A - A + A \times A - A \xrightarrow{R_5} 1 \times \underline{A} - A + A \times A - A \xrightarrow{R_5} \ldots \xrightarrow{R_5} 1 \times 2 - 3 + 4 \times 5 - 6$

Another representation of a derivation is a parse tree. The derivation above is equivalent to the following parse tree,



a. Will the following expressions be evaluated correctly on the embedded system, $w_1 = 1 - 2 + 3 \times 4 \times 6 + 7$, $w_2 = 1 \times 2 \times 3 - 4 + 5 - 6$? If so, prove it by showing a derivation or a parse tree using the rules in AAE.

b. Design a dynamic programming algorithm for this Expression Validation problem which takes as input an arithmetic expression, w, and outputs *true* if w can be generated by AAE, and otherwise outputs *false*, in $O(n^3)$ time.

c. Prove the correctness of your algorithm. As an indication, we expect that the proof will typically be about $\frac{1}{2}$ of a page long.

d. Prove that the running time of your algorithm is $O(n^3)$. As an indication, we expect that the proof will typically be about $\frac{1}{4}$ of a page long.

e. Design a dynamic programming algorithm for **any** Expression Validation problem which takes as input an arithmetic expression, w, and arbitrary rules, AAE', and outputs *true* if w can be generated by AAE', else outputs *false*. Discuss the running time of this more general algorithm. No proofs required.

Problem 3

You are to build a scheduling algorithm that inputs jobs $1, 2, \ldots, n$ each with integer values for duration $t(i)$, deadline $d(i)$, and profit $p(i)$ and outputs a schedule of a **subset** of the jobs run in serial on a machine (where the first job starts at time 0). Not every job must be included in the schedule, but every included job must be completed by its deadline. The goal is to maximize the sum of the profits of jobs that are scheduled.

a. Your boss has suggested three greedy algorithms for this problem.

   (1) Ordered by increasing duration

   (2) Ordered by decreasing profit

   (3) Ordered by increasing deadline

   Each algorithm sorts the jobs by one of the heuristics above then schedules the jobs one by one in that order. Each job is started at the earliest available time and is scheduled only if it will complete before its deadline; Otherwise the job is skipped. Provide counter examples to the alogorithms above to convince your boss that a dynamic program is necessary. Present your counter examples in a table format as follows,

i	duration $t(i)$	deadline $d(i)$	profit $p(i)$
1	1	4	1
2	1	4	1
3	2	4	2

b. Show that, there always exists an optimal solution where the included jobs are scheduled by order of increasing deadline. (Note this is not the same as heuristic 3 in part a)

c. Design a dynamic programming algorithm for the Job Scheduling problem which takes as input the jobs $1, \ldots, n$ and outputs the maximum profit in $O(n \times MaxDeadline)$ time, where $MaxDeadline = Max_{i=1}^{n} d(i)$. Assume that the duration, deadline, and profit functions are defined for each job.

d. Prove the correctness of your algorithm. As an indication, we expect that the proof will typically be about a page long.

e. Prove that the running time of your algorithm is $O(n \times MaxDeadline)$, where $MaxDeadline = Max_{i=1}^{n} d(i)$. As an indication, we expect that the proof will typically be about $\frac{1}{3}$ of a page long.

Problem 4

A carpenter has a piece of wood of a certain length L that must be cut at positions $\{a_1, a_2..., a_n\}$ where a_i is the distance from the left end of the original piece of wood. Notice that after making the first cut, the carpenter now has two pieces of wood; after making the second cut, the carpenter has three pieces of wood, etc. Assume that the cost of making a cut in a piece of wood of length ℓ is equal to ℓ, and is the same no matter which position in that piece of wood is being cut. The goal of the carpenter is to minimize his costs while performing all of the cuts.

a. Given a piece of wood of original length $L = 10$ and a set of positions to cut it at $\{a_1, a_2..., a_n\} = \{1, 4, 5, 8\}$, what is the sequence of cuts that minimizes the cost of all of the cuts?

b. Design a dynamic programming algorithm for the Carpentry problem which takes as input the original length L and a set of cuts defined by $\{a_1, a_2..., a_n\}$ and outputs the minimal cost to make all of the cuts in $O(n^3)$ time.

c. Prove the correctness of your algorithm. As an indication, we expect that the proof will typically be about $\frac{1}{2}$ of a page long.

d. Prove that the running time of your algorithm is $O(n^3)$. As an indication, we expect that the proof will typically be only a few lines long.