Due Apr 3, 2:45pm

Instructions: See “Instructions for writing up homework” on the course web page. Don’t forget your name, your TA’s name, and the list of students you worked with. When writing up a dynamic programming algorithm, in addition to your pseudocode, please specify the vector/matrix you are using (e.g., $E(\cdot)$ or $M(\cdot, \cdot)$), the intuitive meaning of each entry (e.g., “$E(i) =$ length of the longest subsequence starting at offset $i$”), and the recurrence relation for the entries of this vector/matrix (e.g., $E(i) = \max\{1 + E(j) : A[i] < A[j]\}$).

Hint: The generic hint for this problem set is “dynamic programming.”

1. (25 pts.) Scheduling again
We’re going to revisit problem 3(c) from HW8. This time you’ll design an algorithm to find the optimum schedule, in the case where the job lengths are small integers.

As a reminder, you are writing a scheduler for a system consisting of two identical machines. There are $n$ jobs. The $i$th job will take $t_i$ seconds, where each $t_i$ is a positive integer. Each job must be assigned to one of the two machines. We care about the final completion time when the last job is finished. Formally, let $S_1$ denote the set of jobs assigned to the first machine, and $S_2 = \{1, 2, \ldots, n\} \setminus S_1$ the set of jobs assigned to the second machine. The completion time for the first machine is $C_1 = \sum_{i \in S_1} t_i$, and the completion time for the second machine is $C_2 = \sum_{i \in S_2} t_i$. The overall completion time is $C = \max(C_1, C_2)$, and we’d like to find an assignment of jobs to machines that makes $C$ as small as we can.

(a) Design an efficient algorithm to find the cost of the optimal schedule, i.e., the smallest value of $C$ attainable by some assignment of jobs to machines. The running time of your algorithm should be at most $O(nT)$, where $T = \sum_{i=1}^n t_i$. You don’t need to find the assignment itself.

(b) Suppose each integer $t_i$ is $k$ bits long. Express the running time of your solution to part (a), as a function of $k$ and $n$. Use $O(\cdot)$ or $\Theta(\cdot)$ notation.

2. (25 pts.) Another scheduling problem
You’re running a massive physical simulation, which can only be run on a supercomputer. You’ve got access to two (identical) supercomputers, but unfortunately you have a fairly low priority on these machines, so you can only get time slots on them when they’d otherwise be idle. You’ve been given information about how much idle computing power is available on each supercomputer for each of the next $n$ one-hour time slots: you can get $a_i$ seconds of computation done on supercomputer A in the $i$th hour if your job is running on A at that point, or $b_i$ seconds of computation if it running on supercomputer B at that point.
During each hour your job can be scheduled on only one of the two supercomputers. You can move your job from one supercomputer to another at any point, but it takes an hour to transfer the accumulated data between supercomputers before your job can begin running on the new supercomputer, so a one-hour time slot will be wasted where you make no progress.

So you need to come up with a schedule, where for each one-hour time slot your job either runs on supercomputer A, runs on supercomputer B, or “moves” (to switch which supercomputer it will use in the next time slot). If your job is running on supercomputer A for the \( i-1 \)th hour, then for the \( i \)th hour your only two options are to continue running on A or to “move.” The value of a schedule is the total number of seconds of computation that you get done during the \( n \) hours.

You want to find a schedule of maximal value. Design an efficient algorithm to find the value of the optimal schedule, given \( a_1, \ldots, a_n \) and \( b_1, \ldots, b_n \).

Example: Suppose \( n = 4 \) and the inputs are given by

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>( b_i )</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>15</td>
</tr>
</tbody>
</table>

Then the optimal schedule is A, “move”, B, B. Its value is \( 10 + 0 + 3 + 15 = 28 \).

3. (25 pts.) A game

Consider the following game: the casino has placed a sequence \( S[1..n] \) of dollar bills, of various denominations, in a line on the table. The \( i \)th bill is \( S[i] \) dollars. Two players take turns; during your turn, you can pick up and keep either the first or the last bill remaining in the line. The goal is to collect as much money as possible.

Design two algorithms: (1) a \( O(n^2) \) time algorithm that, given \( S[1..n] \), precomputes some information for use by the second algorithm; (2) an \( O(1) \) time algorithm that, given a configuration that can occur in the resulting game and the precomputed information, identifies the best next move.

Warning: Taking whichever bill is larger is not necessarily the best move!

4. (25 pts.) Image re-sizing

In this problem you will explore an application of dynamic programming to automatic re-sizing of images. We are given a (greyscale) image \( I[1..m][1..n] \) with \( m \) rows of pixels and \( n \) columns; \( I[i][j] \) contains the intensity (greyscale level) of the pixel in the \( i \)th row and \( j \)th column. We want to shrink this to an image with \( m \) rows and \( n-1 \) columns, i.e., one column narrower.

We will do this by deleting a \textit{tear} from \( I \). A \textit{tear} is a sequence of pixels that follows a path from the top of the image to the bottom of the image. More precisely, a tear \( T \) is a sequence of pixels \( T = ((1,x_1),(2,x_2),\ldots,(m,x_m)) \), such that: (1) for each \( i \), \( 1 \leq x_i \leq n \); and, (2) for each \( i \), \(|x_{i+1} - x_i| \leq 1 \). To delete a tear, we delete each pixel in the tear from the image \( I \), so removing a single tear shrinks a \( m \times n \)-pixel image to a \( m \times (n-1) \)-pixel image.

We’d like to choose a tear whose removal will be least noticeable to the human eye. Intuitively, one reasonable idea is to choose pixels whose intensity is similar to that of their neighbors (avoiding removal of sharp edges or other kinds of intricate detail). With this idea in mind, we will consider the cost of deleting pixel \( I[i][j] \) to be

\[
\text{cost}(i,j) = |I[i][j-1] - I[i][j]| + |I[i][j] - I[i][j+1]|,
\]

CS 170, Spring 2009, HW 9
and the cost of a tear $T = ((1, x_1), \ldots, (m, x_m))$ to be

$$\text{cost}(T) = \text{cost}(1, x_1) + \cdots + \text{cost}(m, x_m).$$

Design an efficient algorithm to find a minimal-cost tear, given the image $I[1..m][1..n]$. Your algorithm should run in $O(nm)$ time.

**Comment:** This problem introduces you to a powerful technique that was developed by graphics researchers in 2007, for automatic image re-sizing.

Suppose we have a wide image, and we want to display it on a device with a narrower aspect ratio. We could use “letter-boxing”, shrinking the image equally in both the horizontal and vertical dimension; however, letter-boxing leaves black bars at the top and bottom, so you’re not getting the benefit of the full size of the display device. We could re-scale the image in just the horizontal dimension, but that changes the aspect ratio and might make people look skinny or introduce other artifacts. We could crop the image, but that might delete important parts of the image. None of these options seem entirely satisfactory. The researchers developed a way to reduce the image width by automatically identifying “less important” parts of the image and deleting them, in a way that avoids the artifacts of the other approaches.

Here is an example. Consider the following original image:

Here it is cropped (left) and re-scaled (right):
Notice how the cropped image omits some important parts of the scene, and the re-scaled version makes everything look too tall and skinny. The researchers’ basic approach is to repeatedly find a minimal-cost tear and delete it, until the image is shrunk down an acceptable aspect ratio. Here is the result of applying their technique:

Notice how this mostly preserves the look of the objects in the scene and retains the most important parts of the scene (“smooshing together” background scenery as needed to re-size the image). The researchers developed a number of powerful extensions, including automatic re-sizing of video, and removing of selected objects from photographic images. The technology has now been integrated into Adobe Photoshop, under the name “content-aware image resizing.” Pretty amazing stuff.

Image credits: A. Shamir, S. Avidan.

5. (5 pts.) Optional bonus problem
This is an optional challenge problem, worth 5 bonus points if you solve it. Don’t attempt to solve this one unless you have solved all of the other homework questions; this question is intended to be more challenging than the others.

Find a \( O(n \lg n) \) time algorithm for the longest increasing subsequence problem, using the following approach. Build an efficient data structure to store a map \( f : \{1, 2, \ldots, n\} \to \mathbb{N} \), initially zero everywhere, supporting the following three operations:

1. \textsc{Get}(x): given \( x \), retrieve \( f(x) \);
2. \textsc{Increase}(x, y): given \( x, y \) such that \( y > f(x) \), increase \( f(x) \) to \( y \);
3. \textsc{GetMax}(x): given \( x \), retrieve \( \max(f(x+1), f(x+2), \ldots, f(n)) \).

Each of the three operations should run in at most \( O(\lg n) \) time. Then, use this data structure to solve the longest increasing subsequence problem.