COMS21103: Problem set 4

2015/2016

Remark: Most problem sets for the next few weeks will contain at least one starred problem, which is more challenging. If any of the problems seem unclear, please post a question on the Blackboard discussion board.

1. Let A be an array containing n integers. A subarray of A is a contiguous subset of elements of A. The maximum subarray sum MSS is defined to be the maximum, over all subarrays of A, of the sum of the elements in the subarray. For example, if $A = (1, -2, 3, -1, 2)$, $MSS = 4$, achieved by the subarray $(3, -1, 2)$.

   (a) Let $MSS(j)$ be the maximum subarray sum that can be achieved by any subarray of A that finishes at position j. Give a recurrence expressing $MSS(j)$ in terms of $MSS(j-1)$.

   (b) Use your recurrence to give a memoized recursive dynamic programming algorithm for computing MSS. Your algorithm should run in time $O(n)$.

   (c) Give an iterative algorithm which computes MSS without any recursive calls.

2. Suppose that you are in charge of placing posters for the next CSS social on the n lamp posts along Woodland Road. The lampposts are conveniently numbered 1 to n. Each lamppost has a distance x_i (measured in meters from the bottom of Woodland Road) and a value v_i (based on the number of student who will see that poster). Your goal is to plan the placement of the posters to maximise the total value, denoted Val. This is the sum of the values of the lampposts that you put posters on. While there is no explicit limit on the number of posters you can place, according to Council regulations, you cannot place two posters within 10 meters of each other.

   (a) Let $Val(i)$ be the maximum total value that can be achieved by only considering the first i lampposts. Give a recurrence expressing $Val(i)$ in terms of smaller subproblems.

   (b) Use your recurrence to give a memoized recursive dynamic programming algorithm for computing Val. Your algorithm should run in time $O(n)$.

   (c) Give a iterative algorithm which computes Val without any recursive calls.

3. A sequence is a palindrome if it reads the same forwards as backwards. For example the sequence (A, B, C, C, B, A) is a palindrome. The following sequences are also palindromes (A, B, A), (D, E, E, D), and (A). The sequence (A, B, C) is not a palindrome.

   Given a sequence S (of length n), the palindromic subsequence problem is to find the length of the longest subsequence of S which is a palindrome. For example in the sequence $S = (D, A, B, C, A)$ the longest palindromic subsequence is (A, B, A) so the output should be 3.

   Notation: A subsequence of S is any sequence which can be obtained from S by deleting elements (but not rearranging them). The sequences (A, A) and (D, C, A) are both subsequences of (D, A, B, C, A). The sequence (C, B) is not a subsequence of (D, A, B, C, A).

(a) Formulate a recurrence for the palindromic subsequence problem.

(b) Use your recurrence to give a memoized recursive dynamic programming algorithm for the palindromic subsequence problem.

(c) What is the time complexity of your memoized algorithm?

(d) Give a iterative algorithm which solves the palindromic subsequence problem without any recursive calls.

(e) What is the time complexity of your iterative algorithm?

4. ($\star$) The Solitary Drinking Problem is defined as follows. There are n seats in a line at a bar, and a sequence of drinkers comes in to use them. The first drinker sits in seat 1. All subsequent drinkers sit in a seat which is as far as possible from anyone else (if there is more than one, picking the lowest-numbered one). To avoid having to make small talk, drinkers do not want to sit next to anyone else. When this can no longer be achieved, any new drinkers give up and leave.

   For example, with 7 seats and numbering drinkers in order of appearance, the situation at the end of this process, when no more drinkers can sit down, is as follows:

   For a bar with n seats, let $B(n)$ be the number of filled seats at the end of this process. The first values of $B(n)$, starting from $n = 1$, are $1, 1, 2, 2, 3, 3, 3, \ldots$

   Give a dynamic programming algorithm which efficiently computes $B(n)$. Code up your algorithm and use it to compute $B(1000000)$.

For more practice, there are many more dynamic programming exercises by Jeff Erickson online at `http://www.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/05-dynprog.pdf` (I would start with question 3a).