6.006 Introduction to Algorithms
Spring 2008

Introduction to Algorithms: 6.006
Massachusetts Institute of Technology
Professors Srini Devadas and Erik Demaine

May 9, 2008
Handout 13

Final Practice Problems

1 Subset Sum

You are given a sequence of n numbers (positive or negative):

$$x_1, x_2, \ldots, x_n$$

Your job is to select a subset of these numbers of maximum total sum, subject to the constraint that you can't select two elements that are adjacent (that is, if you pick x_i then you cannot pick either x_{i-1} or x_{i+1}).
Explain how you can find, in time polynomial in n, the subset of maximum total sum.

Solution: Let sum_i be the maximum sum of the numbers $x_1, x_2, \ldots, x_i$ given the adjacency constraint.

$$\begin{aligned} sum_0 &= 0 \\ sum_1 &= \max(0, x_1) \\ sum_i &= \max(sum_{i-2} + x_i, sum_{i-1}) \end{aligned}$$

This last step works because either we include x_i, in which case we also want to include the best solution on up to $i-2$, or we don't include x_i, in which case we can just use the best solution on $i-1$.
Our final answer is then just sum_n.
To calculate the set that gives the max sum, we could simply keep pointers back from i to either $i-1$ or $i-2$ depending on which one was bigger (or we could go back and check which was bigger). We follow those pointers, including appropriate numbers.
Because there are n subproblems, and each subproblem takes $O(1)$ time to solve, this runs in $O(n)$ time.

2 Collecting Coins

You are given an n-by-n grid, where each square (i, j) contains $c(i, j)$ gold coins. Assume that $c(i, j) \geq 0$ for all squares. You must start in the upper-left corner and end in the lower-right corner, and at each step you can only travel one square down or right. When you visit any square, including your starting or ending square, you may collect all of the coins on that square. Give an algorithm to find the maximum number of coins you can collect if you follow the optimal path.

Solution: We will solve the following subproblems: let $dp[i, j]$ be the maximum number of coins that it is possible to collect while ending at (i, j).
We have the following recurrence:

$$dp[i, j] = c(i, j) + max(dp[i - 1, j], dp[i, j - 1])$$

We also have the base case that when either $i = 0$ or $j = 0$, $dp[i, j] = c(i, j)$.
There are n^2 subproblems, and each takes $O(1)$ time to solve (because there are only two subproblems to recurse on). Thus, the running time is $O(n^2)$.

3 True/False

Decide whether these statements are **True** or **False**. You must briefly justify all your answers to receive full credit.

1. Any Dynamic Programming algorithm with n subproblems will run in $O(n)$ time.
   True **False**

   Explain:

   Solution: **False.** The subproblems may take longer than constant time to compute, as was the case with with longest increasing subsequence.

2. Karatsuba's method is based on the use of continued fractions.
   True **False**

   Explain:

   Solution: **False.** Karatsuba has nothing to do with continued fractions.

3. Newton's Method for computing $\sqrt{2}$ essentially squares the number of correct digits at each iteration.
   True **False**

   Explain:

   Solution: **False.** Newton's method for computing $\sqrt{2}$ has quadratic convergence, which means the number of correct digits doubles after each iteration.

4 Numerics

Suppose we are trying to compute $\sqrt[3]{9}$ (the cube root of 9).

Explain carefully how one iteration of Newton's Method works for this problem, starting with an initial guess of $x_0 = 2$. (Hint: the function to use is $f(x) = x^3 - 9$.) Be sure to derive carefully the value of x_1.

Solution:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$
$$x_1 = x_0 - \frac{x_0^3 - 9}{3 * x_0^2}$$
$$x_1 = 2 - \frac{2^3 - 9}{3 * 2^2}$$
$$x_1 = 2 - \frac{-1}{12}$$
$$x_1 = \frac{25}{12}$$