Problem Set 6

This final problem set explores dynamic programming in several contexts. We hope that this gives you a feel for the breadth of applications where DP solutions are appropriate.

Please be sure to write your answers different problems on separate pieces of paper to make it easier for us to grade. Also, **please put your name on each page of your assignment**.

As always, please feel free to drop by office hours or send us emails if you have any questions. We'd be happy to help out.

This problem set has 29 possible points. It is weighted at 12% of your total grade.

Good luck, and have fun!

Due Monday, August 12 at 2:15 PM

Note: You may submit this problem set on Wednesday, August 14 for full credit if you would like, but be aware that the final project will be distributed on Monday, August 12 and you may want to budget your time appropriately. We've tried to make this problem set a bit shorter than the others so that you can have some time to begin reviewing for the final project.

Problem One: Coin Changing (8 Points)

Suppose that you are living in a country where coins have denominations $¤c_1 < ¤c_2 < \ldots < ¤c_n$ (here, ¤ is some arbitrary prefix for a currency; it could be \$, €, ¥, etc.) You are interested in determining the fewest number of coins necessary to total to exactly ¤*N*.

In most currency systems, you can use a greedy algorithm to find the fewest number of coins necessary to express ¤*N* by greedily selecting the largest coin you can add to the collection without exceeding ¤*N* until ¤*N* is reached. For example, to make 74¢ with US coins, you would add a half-dollar (total 50¢), two dimes (total 70¢), and four pennies (total 74¢). However, this does not work in all currency systems. For example, in a currency system where coins have denominations ¤11, ¤5, and ¤1, the greedy algorithm for finding coins totaling ¤15 would require five coins: one ¤11 coin and four ¤1 coins. However, the optimal solution requires only three ¤5 coins.

Design an algorithm for finding the smallest number of coins necessary to represent ¤*N*, given that the currency has coins of *n* denominations $¤c_1 < ¤c_2 < \ldots < ¤c_n$. Your algorithm just needs to give back the *number* of coins necessary to do this, rather than the specific collection of coins you would use. Your algorithm should run in time polynomial in *n* and *N*. Then:

- Describe your algorithm.
- Prove your algorithm returns the minimum number of coins necessary to represent ¤*N*.
- Prove your algorithm runs in time polynomial in *n* and *N*.

In writing your algorithm, you can assume that one of the coins is a ¤1 coin so that it is possible to make change for all possible denominations.

Problem Two: Parenthesizing Expressions (10 Points)

Suppose that you are given an unparenthesized mathematical expression containing *n* numbers, where the only operators are + and –, as shown here:

$$4 + 3 – 2 – 5 + 1 – 6 + 7$$

You can change the value of the expression by adding parentheses in different positions. For example:

$$4 + 3 – 2 – 5 + 1 – 6 + 7 = 2$$

$$(4 + 3 – (2 – 5)) + (1 – 6) + 7 = 12$$

$$(4 + (3 – 2)) – (5 + 1) – (6 + 7) = –14$$

For the purposes of this problem, we'll assume you can only put parentheses immediately before and immediately after a number, so that you couldn't interpret the expression multiplicatively as

$$4 + 3 (–2) (–5) + 1 – 6 + 7 = 36$$

Design an algorithm that, given an unparenthesized expression using only + and –, determines the maximum possible value the expression can take when parentheses are added in. Your algorithm should run in time polynomial in *n*. Then:

- Describe your algorithm.
- Prove that your algorithm finds the maximum possible value the expression can take.
- Prove that your algorithm runs time polynomial in *n*.

Problem Three: Cell Towers Revisited (10 Points)

In lecture, we discussed the problem of placing cell towers in a line of towns to cover the maximum amount of people. If you'll recall, we were prohibited from placing cell towers in two adjacent cities because the signals from the cell towers would interfere with one another. Additionally, each cell tower provided coverage only to the town in which it was placed.

Let's now suppose that you've upgraded your cell towers to a newer version that has neither of these restrictions. Specifically, these cell towers don't interfere with one another when placed in adjacent towns (so you can place them wherever you'd like), and they transmit with enough power to provide coverage to the town in which they're built and to adjacent towns. With towers like these, it's always possible to provide coverage to all the towns.

The challenge now is to determine the cheapest possible way to build cell towers that cover everyone. Suppose that you have a cost c_i associated with building a cell tower in town c_i. Your job is to determine the minimum cost of cell towers necessary to cover all of the towns. For example, suppose the towns are arranged in this linear order with their costs indicated:

$314	$159	$265	$358	$979	$323	$846	$264	$338
Town 1	Town 2	Town 3	Town 4	Town 5	Town 6	Town 7	Town 8	Town 9

An expensive way to provide coverage to all towns would be to buy a cell tower in each town. A better option would be to build cell towers in the first, third, fifth, etc. towns, as shown here:

$314	$159	$265	$358	$979	$323	$846	$264	$338
✓		✓		✓		✓		✓

An even better option would be to purchase these towers:

$314	$159	$265	$358	$979	$323	$846	$264	$338
	✓		✓		✓		✓	

An even better option would be to purchase these towers:

$314	$159	$265	$358	$979	$323	$846	$264	$338
	✓	✓			✓		✓	

Design an algorithm that finds the minimum cost necessary to provide cell coverage to all of the towns. Note that you just need to find the cost of the necessary cell towers and don't need to report which towers you will purchase. Your algorithm should run in time polynomial in n, where n is the number of towns. Then:

- Describe your algorithm.
- Prove your algorithm finds the minimum cost necessary to provide cell coverage to all of the towns.
- Prove your algorithm runs in time polynomial in n.

(Hint: Consider having different types of subproblems for the case where the towns on the boundary are already covered and where the towns on the boundary still need to be covered.)

Problem Four: Course Feedback (1 Point)

We want this course to be as good as it can be, and we'd appreciate your feedback on how we're doing. For a free point, please answer the following questions. We'll give you full credit no matter what you write, as long as you write something.

i. How hard did you find this problem set? How long did it take you to finish? Does that seem unreasonably difficult or time-consuming for a five-unit course?

ii. Did you attend office hours? If so, did you find them useful?

iii. Did you read through the textbook? If so, did you find it useful?

iv. How is the pace of this course so far? Too slow? Too fast? Just right?

v. Is there anything in particular we could do better? Is there anything in particular that you think we're doing well?