COMP 550.001 - Fall 2017
Assignment 3

Part A due: Wednesday, September 27, 2017 (start of class)
Part B due: Friday, September 29, 2017 (4:00 p.m.)

For Part A, you should submit a physical copy of your written homework at the start of class.
For Part B, you should submit a .tar.gz or .zip file with your solutions on Sakai.

Optional Part C due: Friday, September 29, 2017 (start of class)

This assignment includes an optional Part C. Earning half of the points will be worth half of a late day (only integral late days may be used to turn in homework late, but a partial late day can count as partial extra credit at the end of the semester), and earning at least 80% of the points will be worth a full late day. You should submit your code in a .zip or .tar.gz file to Sakai, and the analysis in a physical copy.

Part A: Due Wednesday, September 27, 2017

Be sure to include a collaboration statement with your assignment, even if you worked alone.

[16 points] Problem 1: CLRS Exercise 15.1-2

Show, by means of a counterexample, that the following "greedy" strategy does not always determine an optimal way to cut rods. Define the ***density*** of a rod of length i to be p_i/i, that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where $1 \leq i \leq n$, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length $n - i$.

[8 points] Problem 2: CLRS Exercise 15.2-1

Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is $\langle 5, 10, 3, 12, 5, 50, 6 \rangle$. Be sure to show your work.

[12 points] Problem 3: CLRS Exercise 15.3-2

Draw the recursion tree for the MERGE-SORT procedure from Section 2.3.1 on an array of 16 elements. Explain why memoization fails to speed up a good divide-and-conquer algorithm such as MERGE-SORT.

[24 points] Problem 4: CLRS Problem 15-2

A ***palindrome*** is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `civic`, `racecar`, and `aibohphobia` (fear of palindromes).

Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input `character`, your algorithm should return `carac`. What is the running time of your algorithm?

Part B: Due Friday, September 29, 2017

[40 points] Problem 1

For this problem, you will write a series of unit tests for the matrix-chain multiplication problem introduced in lecture. You are given working dynamic programming implementation of this algorithm in `calculateMatrixChainOrder`.

The units tests you'll be writing are what are known as *white-box tests*. This means that you should write them with the actual implementation in mind.

Unit tests are often divided into two separate categories: *happy-path tests*, which check for the correct computation of results given valid inputs, and tests that check for invalid input, e.g. `null` parameters, empty lists, etc.

You will be graded on *code coverage*. This means that to get full points, you must test every possible path of the code. For example, if there is an if statement, like the following:

if $n < 0$ **or** $n > 7$:
    return 0
else:
    return 2

then you should have three unit tests, one for a negative value of n, with an expected result of 0, one for some value of n that is greater than 7, with an expected value of 0, and one for some value of n that is at least 0 and less than 8, with an expected result of 2.

You can test code coverage using the EclEmma Java Code Coverage plugin in Eclipse[1]. Code coverage is typically depicted in two ways. First, after running your tests, lines that are completely executed are colored green, and lines that are completely missed are colored red. Lines that are partially executed, such as only checking for $n < 0$ in the example above, are colored yellow. This is usually the case due to if statements, especially with short-circuiting. Second, you can view code coverage on a percentage basis. This usually gives you the total count and percentage of lines/instructions/etc. that were covered and missed during the execution.

Although verbose, a good structure for unit tests is one that is easy to read. Here is my general structure:

```
// 1. Declare variables needed to set up the test
// 2. Set up the expected results
// 3. Call the function
// 4. Verify expected output
```

Two example unit tests are provided for you. You should supplement them with your own.

---

[1] `http://crunchify.com/what-is-the-best-code-coverage-plugin-you-should-use-in-eclipse-ide/` gives a reasonable tutorial for getting EclEmma installed and running.

[Optional] Part C: Due Friday, September 29, 2017

Variant of CLRS Problem 15-8: Image Compression via Seam Carving

We are given a color picture consisting of an $m \times n$ array $A[1..m, 1..n]$ of pixels, where each pixel specifies a triple of red, green, and blue (RGB) intensities. Suppose that we wish to compress this picture slightly. Specifically, we wish to remove one pixel from each of the m rows, so that the whole picture becomes one pixel narrower. To avoid disturbing visual effects, however, we require that the pixels removed in two adjacent rows be in the same or adjacent columns; the pixels removed form a "seam" from the top row to the bottom row where successive pixels in the seam or adjacent vertically or diagonally.

This algorithm is described in this YouTube video: `https://youtu.be/vIFCV2spKtg`.

a.

Show that the number of such possible seams grows at least exponentially in m, assuming that $n > 1$.

b.

Suppose now that along with each pixel $A[i, j]$, we have calculated a real-valued disruption measure $d[i, j]$, indicating how disruptive it would be to remove pixel $A[i, j]$. Intuitively, the lower a pixel's disruption measure, the more similar the pixel is to its neighbors. Suppose further that we define the disruption measure of a seam to be the sum of the disruption measures of its pixels.

Give an algorithm to find a seam with the lowest disruption measure. How efficient is your algorithm?

c.

Implement your algorithm by filling in the `findBestSeam` method in the provided skeleton code.

Part C is derived from MIT's 6.006 course, Fall 2011:

- Erik Demaine and Srinivas Devadas, 6.006 Introduction to Algorithms, Fall 2011. (MIT OpenCourseWare: Massachusetts Institute of Technology): https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/index.htm (Accessed September 15, 2017). License: Creative commons BY-NC-SA