Homework 3: Dynamic Programming, Greedy
DUE: 11:30pm Mon, Mar 22, 2021, on Blackboard

- Please type and submit PDF file to Blackboard. No late submissions will be accepted.
- **Academic honesty**: at the beginning of your paper, please include: "I understand and agree to abide by the provisions in the University of Houston Undergraduate Academic Honesty Policy" and "This is my own work."
- Discussion is encouraged. In each problem solution, list the names of persons that you have discussed with about that problem, except for large group discussion such as in MS Teams where a note of "group discussion" suffices. If an online source helped you, list the URL.
- **Unless otherwise statement, for an algorithm design problem, please include algorithm description (texts + pseudocode), proof of correctness, and analysis of worst case time complexity (big O or big Theta)**.

Dynamic Programming:

P0 (20pt): This exercise asks you to develop efficient algorithms to find optimal *subsequence* of various kinds. A subsequence is anything obtained from a sequence by extracting a subset of elements, but keeping them in the same order; the elements of the subsequence need not be contiguous in the original sequence. For example, the strings `C`, `DAMN`, `YAIOAI`, and `DYNAMICPROGRAMMING` are all subsequences of the string `DYNAMICPROGRAMMING`.

a) Let $A[1..m]$ and $B[1..n]$ be two arbitrary arrays. A *common subsequence* of A and B is another sequence that is a subsequence of both A and B. Describe an efficient algorithm to compute the length of the **longest common subsequence** of A and B.

b) Call a sequence $X[1..n]$ of numbers *bitonic* if there is an index i with $1<i<n$, such that the prefix $X[1..i]$ is increasing and the suffix $X[i..n]$ is decreasing. Describe an efficient algorithm to compute the length of the **longest bitonic subsequence** of an arbitrary array X of integers.

c) Call a sequence $X[1...n]$ of numbers *oscillating* if $X[i]<X[i+1]$ for all even i, and $X[i]>X[i+1]$ for all odd i. Describe an efficient algorithm to compute the length of the **longest oscillating subsquence** of an arbitrary array X of integers.

P1 (20pt). Describe and analyze an algorithm that finds the maximum-area rectangular pattern that appears more than once in a given bitmap. Specifically, given a two-dimensional array $M[1..n, 1..n]$ of bits as input, your algorithm should output the area of the largest repeated rectangular pattern in M. For example, given the bitmap shown on the left in the figure below, your algorithm should return the integer 195, which is the area of the 15×13 doggo. (Although it doesn't happen in this example, the two copies of the repeated pattern might overlap.) For full credit, describe an algorithm that runs in $O(n^5)$ time. For partial credit, the algorithm must run in $O(n^6)$ time.

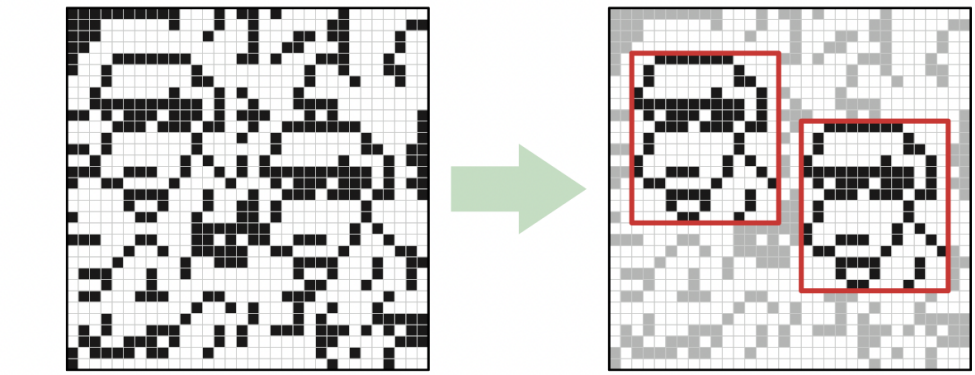

Figure. In the pixel map, the 15×13 doggo square appear twice.

P2 (20pt): Suppose you are given a sequence of integers separated by $+$ and $-$ signs; for example:

$$1+3-2-5+1-6+7$$

You can change the value of this expression by adding parentheses in difference places. For example:

$$\begin{aligned} 1+3-2-5+1-6+7 &= -1 \\ (1+3-(2-5))+(1-6)+7 &= 9 \\ (1+(3-2))-(5+1)-(6+7) &= -17 \end{aligned}$$

Describe and analyze an algorithm to compute, given a list of integers separated by $+$ and $-$ signs, the maximum possible value the expression can take by adding parentheses. Parentheses must be used only to group additions and subtractions; in particular, do not use them to create implicit multiplication as in 1 + 3(-2)(-5) + 1- 6 + 7 = 33.

P3 (20pt) A string w of parentheses (and) and brackets [and] is balanced if it satisfies one of the following conditions:

- w is the empty string
- $w = (x)$ for some balanced string x
- $w = [x]$ for some balanced string x
- $w = xy$ (concatenation) for some balanced strings x and y

For example, the string
w=([()][]())[()()]() is balanced, because $w=xy$, where x=([()][]()) and y=[()()]()

1. Describe and analyze an algorithm to determine whether a given string of parentheses and brackets is balanced.
2. Describe and analyze an algorithm to compute the length of a longest balanced subsequence of a given string of parentheses and brackets.

Greedy algorithms: Please provide complete proof of correctness! The proof usually involves the exchange argument. You don't have to prove you are correct if you are not using greedy algorithm.

P4 (20pt) Let X be a set of n intervals on the real line. We say that a subset of intervals $Y \subseteq X$ covers X if the union of all intervals in Y is equal to the union of all intervals in X . The size of a cover is just the number of intervals.

Describe and analyze an efficient algorithm to compute the smallest cover of X. Assume that your input consists of two arrays $L[1..n]$ and $R[1..n]$, representing the left and right endpoints of the intervals in X.

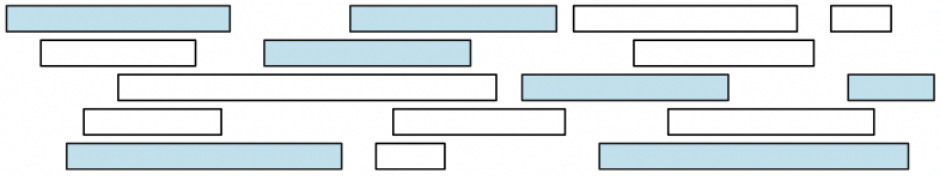

Figure 1. Example of a cover. This is not the smallest cover.