ECE 374 B - Fall 2023

Homework 5

- **Submit your solutions electronically on the course Gradescope site as PDF files.** If you plan to typeset your solutions, please use the LaTeX solution template on the course web site. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app (or an actual scanner, not just a phone camera). We will mark difficult to read solutions as incorrect and move on.
- **Every homework problem must be done *individually*.** Each problem needs to be submitted to Gradescope before 6AM of the due data which can be found on the course website: https://ecealgo.com/homeworks.html.
- For nearly every problem, **we have covered all the requisite knowledge required to complete a homework assignment prior to the "assigned" date.** This means that there is no reason not to begin a homework assignment as soon as it is assigned. Starting a problem the night before it is due a recipe for failure.

Policies to keep in mind

- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use, and you ***must*** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- **Being able to clearly and concisely explain your solution is a part of the grade you will receive.** Before submitting a solution ask yourself, if you were reading the solution without having seen it before, would you be able to understand it within two minutes? If not, you need to edit. Images and flow-charts are very useful for concisely explain difficult concepts.

See the course web site (https://ecealgo.com) for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Piazza.

Extra Instructions Solutions to a dynamic programming problem have (at minimum) three things:

- A recurrence relation
- A *brief* description of what your recurrence function represents and what each case represents.
- A *brief* description of the memory element/storage and how it's filled in.

---

1. Suppose we have a river and on either side are a number of cities numbered from 1 to n (North side: $N[1 \ldots n]$, South side: $S[1 \ldots n]$). The city planner wants to connect certain cities together using bridges and has a list of the desired crossings (x is a $2 \times k$ array where k is the number of planned bridges) . Unfortunately, as we know, bridges cannot cross one-another over water so the city planner must focus on building the most bridges from his plan that do not intersect. Describe an algorithm that finds the maximum number of non-intersecting bridges.

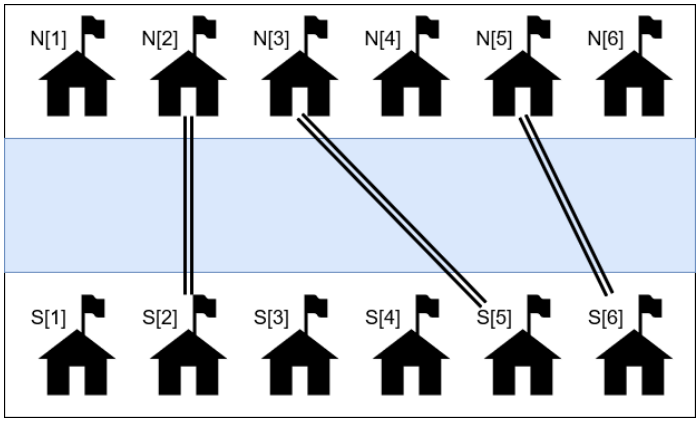


Figure 1. Assuming $n = 6$, $x = \begin{bmatrix} 1 & 5 & 6 & 2 & 3 \\ 4 & 6 & 1 & 2 & 5 \end{bmatrix}$, then the output should be 3 as shown above.

2. In lecture we defined the recurrence of the longest-increasing-subsequence(*LIS*) problem as:

$$LIS_{LEC}(i,j) = \begin{cases} 0 & i = 0 \\ LIS_{LEC}(i-1,j) & A[i] \geq A[j] \\ \max \begin{cases} LIS_{LEC}(i-1,j) \\ 1 + LIS_{LEC}(i-1,i) \end{cases} & A[i] < A[j] \end{cases} \tag{1}$$

But when we worked out the problem in lab looks like:

$$LIS_{LAB}(i,j) = \begin{cases} 0 & \text{if } i > n \\ LIS_{LAB}(i+1,j) & \text{if } i \leq n \text{ and } A[j] \geq A[i] \\ \max \left\{ \begin{array}{c} LIS_{LAB}(i+1,j) \\ 1 + LIS_{LAB}(i+1,i) \end{array} \right\} & \text{otherwise} \end{cases} \tag{2}$$

Is one of them wrong? If not, what's the difference? Your solution should be a simple, **short**, english description of each recurrence. No long proofs for correctness are necessary. This is to make sure you understand how to describe a function (and no, saying "LIS returns the longest increasing subsequence length." is not a sufficient description).

3. A *subsequence* is any sequence obtained from a sequence by taking some of its elements while keeping them in the same order. For instance, the strings 7, $F4$ and $WTF374$ are all subsequences of the string $WTF374$. A *supersequence* is any sequence obtained from a sequence by adding more elements, keeping the elements of the original sequence in the same order. For example, $WTF374$, $SMTWTFS247374$ and $WTF374473FTW$ are all supersequences of the string $WTF374$. For each part, we want efficient (i.e., polynomial time) algorithms.

   (a) Suppose $X[1..m]$ and $Y[1..n]$ are two arrays. A *common subsequence* of X and Y is another sequence that is a subsequence of both X and Y. Describe an algorithm to compute the *length* of a longest common subsequence of two given arbitrary arrays A and B.

   (b) Suppose $X[1..m]$ and $Y[1..n]$ are two arrays. A *common supersequence* of X and Y is a sequence that contains both X and Y as subsequences. Describe an algorithm to compute the *length* of a shortest common supersequence of two given arbitrary arrays A and B.

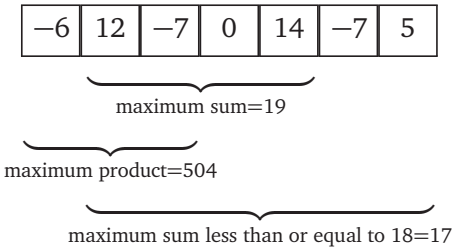
   (c) A sequence $W[1..n]$ of numbers is *weakly increasing* if each element is larger than the average of its two previous elements (i.e., $2 \cdot W[i] > W[i-1] + W[i-2]$ for all $i > 2$). Describe an algorithm to compute the *length* of a longest weakly increasing subsequence of a given arbitrary array A of integers.

   (d) A sequence $O[1..n]$ of numbers is *oscillating* if $O[i] > O[i+1]$ for all odd i and $O[i] < O[i+1]$ for all even i. Describe an algorithm to compute the *length* of a shortest oscillating supersequence of a given arbitrary array A of integers.

4. Suppose you are given an array $A[1..n]$ of arbitrary real numbers. Recall a *subarray of an array A* is by definition a *contiguous* subsequence of A. Define the sum and product of an empty array to be 0 and 1, respectively. For any array $A[i..j]$ where $i \le j$, define its sum and product to be

$$\sum_{k=i}^{j} A[k] \quad \text{and} \quad \prod_{k=i}^{j} A[k],$$

respectively. For the sake of analysis, assume that comparing, adding and multiplying any pair of numbers takes $O(1)$ time.



(a) Describe and analyze an algorithm to compute the *maximum sum* of any subarray of A. For example, given $A = [-6, 12, -7, 0, 14, -7, 5]$, your algorithm should return 19 as illustrated above.

(b) Describe and analyze an algorithm to compute the *maximum product* of any subarray of $A[1..n]$. For example, given $A = [-6, 12, -7, 0, 14, -7, 5]$, your algorithm should return 504 as illustrated above.

(c) Suppose you are also given an arbitrary integer $X \ge 0$. Describe and analyze an algorithm to compute the *maximum sum* of any subarray of *A less than or equal to X*. For example, given $A = [-6, 12, -7, 0, 14, -7, 5]$ and $X = 18$, your algorithm should return 17 as illustrated above.

(d) Describe a faster algorithm for part (c) when every element in the array A is nonnegative.