ECE 374 B ✧ Spring 2024

Homework 5

- **Submit your solutions electronically on the course Gradescope site as PDF files.** 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).

☞ Some important course policies ☜

- **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.
- **Avoid the Three Deadly Sins!** Any homework or exam solution that breaks any of the following rules will be given an ***automatic zero***, unless the solution is otherwise perfect. Yes, we really mean it. We're not trying to be scary or petty (Honest!), but we do want to break a few common bad habits that seriously impede mastery of the course material.
- For algorithmic problems, **we will be grading clarity and conciseness of solutions** (in addition to correctness) which admittedly is a subjective measure, but it is necessary. Being able to communicate your code is a far more important skill than being able to code itself.
- 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.
- Last minute tips:
  - Always give complete solutions, not just examples.
  - Always declare all your variables, in English. In particular, always describe the specific problem your algorithm is supposed to solve.
  - Never use weak induction.

See the course web site 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.

Note: Be aware of new instructions for DP problems on the first page.

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. You may assume that a city would not appear more than once in the plan.

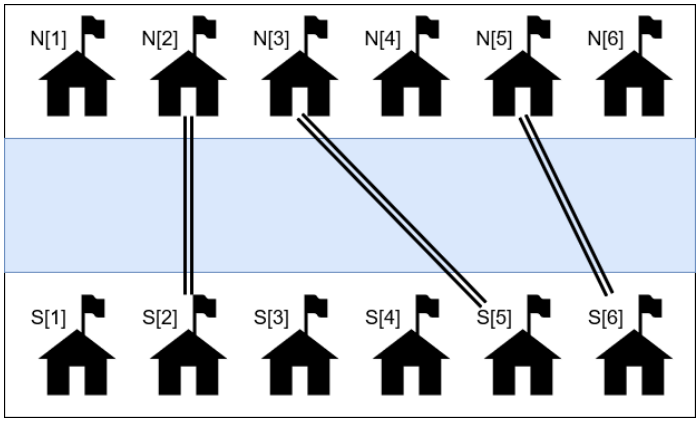

Figure 1. Assuming $n = 6$, $x = \begin{bmatrix} 1 & 5 & 6 & 2 & 3 \\ 4 & 6 & 1 & 2 & 5 \end{bmatrix}$, then the output should be 3 as shown above.

2. For each of the following recurrences, provide English description in terms of the parameters i and j.

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

   (b)
   $$LIS_{LAB}(i,j) = \begin{cases} 0 & \text{if } i > n \\ LIS_{LAB}(i+1,j) & \text{if } i \leq n \text{ and } A[j] \geq A[i] \\ \max \begin{Bmatrix} LIS_{LAB}(i+1,j) \\ 1 + LIS_{LAB}(i+1,i) \end{Bmatrix} & \text{otherwise} \end{cases}$$

   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. Given an $n \times m$ grid filled with non-negative numbers, find the minimal sum of all numbers along a path from top left $(1,1)$ to bottom right (n,m). You can only move either down $(++i)$ or right $(++j)$ at any point in time. Find an algorithmically efficient solution. What is the running time of your algorithm?

4. A certain string processing language allows the programmer to break a string into two pieces. It costs n units of time to break a string of n characters into two pieces, since this involves copying the old string. A programmer wants to break a string into many pieces, and the order in which the breaks are made can affect the total amount of time used. For example, suppose we wish to break a 20-character string after characters 3, 8, and 10. If the breaks are made in left-to-right order, then the first break costs 20 units of time, the second break costs 17 units of time, and the third break costs 12 units of time, for a total of 49 units. If the breaks are made in right-to-left order, the first break costs 20 units of time, the second break costs 10 units of time, and the third break costs 8 units of time, for a total of only 38 units.

   Give a dynamic programming algorithm that takes a list of character positions after which to break and determines the cheapest break cost in $O(n^3)$ time.