Homework 1

- **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/su24/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/su24/) 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.

1. **Herding a Dumb Horse Across a Chessboard** One is interested in the paths of the knight (according to the chess game) over a board with $n > 4$ rows and $m > 3$ columns. One wants to know the number of distinct ways to go from the square $(1, 1)$ (departure) to the square (n, m) (arrival). By convention, (i, j) stands for the square of the board located on row i and column j. In contrast to the chess game where the knight has eight possible moves (except if it would go outside the board), it is imposed that the knight moves only in order to increase the column index (the second one), as shown in Figure 1.

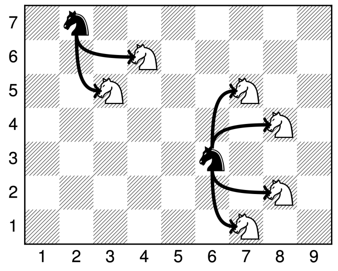

Figure 1. The knight located in $(7, 2)$ (respectively $(3, 6)$), plotted in black, can move to the two (respectively four) positions, plotted in white.

Let nbRout(i, j) be the count of distinct paths starting in $(1, 1)$ and ending in (i, j). Give the complete recurrence for calculating nbRout(i, j).

2. **Don't lift the Da*n Pen!** We want to draw n doubly nested squares as follows.

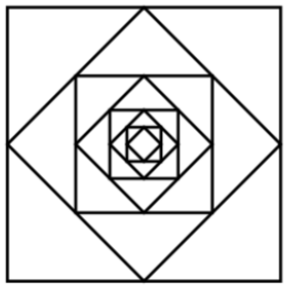

Figure 2. A drawing involving four nested double squares.

- at the end of the drawing, the pen must be back to its starting position,
- the pen must not be raised during the drawing and no segment can be drawn more than once (neither idle time, nor useless work).

a. Identify the possible starting point(s) of the drawing.

b. Write a recursive algorithm to make this drawing.

3. **Array-nageddon: The Ultimate Backtracking Hellscape** Let $T[1..n]$ ($n \geq 1$) be an array of real non-negative values ($T \in 1..n \rightarrow \mathbb{R}^{+}$). There are at least two indices, i and j, defining the interval $i..j$, with $1 \leq i \leq j \leq n$, such that the value of the expression $T[j] - T[i]$ is maximum. We are looking for this maximum value (the value of the best interval). The special case where $i = j$ characterizes a monotonic array strictly decreasing: the value searched for is then zero. Give a recurrence to solve this problem! Make sure to give an English description.

4. **Turning Your Recursion into a Memoization Mess** Describe a memorization order and data structure to convert your answer from to Problem 3 from a recursive backtracking solution into a Dynamic Programming Solution.

5. **Turning Alphabet Soup into a Clusterf*k** Let C be a set of m words on the alphabet Σ, all with lengths less than or equal to k (C is called the code). We also have another word D of length n on the alphabet Σ, which we try to encode using the fewest possible occurrences of words of C. For example, if $C = \{a, b, ba, abab\}$ and $D = babbaababa$, a possible encoding of D is $ba\ ba\ b\ ba$, using six occurrences of C. There may be no solution, as for the encoding of $D = abbc$ with the code $C = \{a, bc\}$. A sufficient condition for any string to be encoded (and thus admit optimal encoding) is that Σ be included (in a broad sense) in C.