Instructions: Please note that handwritten assignments **will not be graded**. Use the provided LaTeX template to complete your homework. Please do not alter the order or spacing of questions (keep each question on its own page). When you submit to Gradescope, you must mark which page(s) correspond to each question. **You may not receive credit for unmarked questions**.
When including graphical figures, we encourage the use of tools such as graphviz or packages like tikz for simple and complex figures. However, these may be handwritten only if they are neat and legible (as defined by the grader).

List any collaborators (besides TAs or professors) here:

1. (55 points) [W9, ★★★★★] The traditional world chess championship is a match of 24 games. The current champion retains the title in case the match is a tie[1]. Each game ends in a win, loss, or draw (tie) where wins count as 1, losses as 0, and draws as $\frac{1}{2}$. The players take turns playing white and black with the champion playing white in the first game.

   Assume that $W(c)$, $D(c)$, and $L(c)$ return the probabilities that the champion wins, draws, or loses *one game* while playing color $c \in \{w, b\}$, respectively[2]. For example, $D(b)$ is the probability that the champion draws while playing black. You won't specifically need to use this, but convince yourself that $W(c) + D(c) + L(c) = 1.0$.

   Your task is to **calculate the champion's probability of retaining the title**.

   For the following questions, use the variable g to represent the number of games remaining in the match, and the variable i to represent the (possibly fractional) number of points that the champion needs to win to retain the title.

   (a) (5 points) What are the inputs to the **subproblem instance**? (**Hint:** your answer will be in terms of the variables described above.)

   (b) (5 points) Which subproblem instances are the **base cases**? What are their values? (**Hint:** think about subproblem instances where the probability of the champion retaining the title are not dependent on any of the win/draw/loss probabilities.)

   (c) (15 points) Given your answers to the above questions, develop notation and write a *formula* the **recurrence relation**. Assume that the function $C(g)$ returns the color the champion is playing in game g.

   (d) (5 points) How many **dimensions** will the dynamic programming table (used in bottom-up DP) have? What will each dimension of your table represent? What is the domain of each dimension?

   (e) (5 points) In **which cell of your bottom-up DP table will you find the answer** to the posed question (the champion's probability of retaining the title)?

   (f) (10 points) What **iteration order** is required to compute the bottom-up DP table?

[1] This is no longer the format used to determine the champion. Instead, sets of progressively shorter games are played until one player has broken the tie.

[2] Chess outcomes tend to be better playing white because white gets the first move.

(g) (10 points) Imagine that the championship match was a variable number of games n. What is the **complexity of this bottom-up DP algorithm in terms of** n?

2. (45 points) [W10, ★★★★] A palindrome is a string that is equal to itself when reversed. Formally, a string $s_1 s_2 \cdots s_n$ is a palindrome if $s_1 s_2 \cdots s_{n-1} s_n = s_n s_{n-1} \cdots s_2 s_1$. According to wikipedia, the 19-letter Finnish word saippuakivikauppias is the longest palindromic word in everyday use.

   Given a string, S, you want to find the **longest palindromic subsequence**. That is, the longest subsequence of S where the subsequence is a palindrome. Remember, a subsequence is *not* necessarily continuous.

   (a) (5 points) Provide two examples of longest palindromic subsequences for the following string: `COMPUTERSCIENCE`.

   (b) (5 points) What are the inputs to the **subproblem instance**?

   (c) (5 points) Which subproblem instances are the **base cases**? What are their values?

   (d) (15 points) Given your answers to the above questions, develop notation and write a *formula* the **recurrence relation**.

   (e) (15 points) Write a **"top-down" DP solution** that augments the recursive algorithm above with memoization. Store intermediate results in a dictionary/array and check to see if a value has already been determined before computing it again. If the value has already been computed, then return it instead of making the recursive call(s). **Present your code using the minted LaTeX library. You shouldn't need more than 10 lines.**