Assignment 3: Dynamic Programming (10%)

Winter 2024 – CPSC 413

Due at 23:59, Mar. 26 on Gradescope

Assignment policy:

- This is an **individual** assignment, so the work you hand in must be your own. Any external sources used must be properly cited (see below).
- Submit your answers to the written questions as a **single PDF** to the correct location on Gradescope.
- See D2L for the assignment late policy. You are responsible for tracking how many flex days you have used. Remember that once these flex days have been used, late assignments **will not be accepted**.
- All submissions must be clearly legible. Handwritten assignments will be accepted, but marks may be deducted if the TAs cannot read your writing. It is recommended that you use LaTeX to typeset your work.
- Some tips to avoid plagiarism in your assignments:

  1. Discuss and share ideas with other students as much as you like, but make sure that when you write your assignment, it is your own work. A good rule of thumb is to wait 20 minutes after talking with somebody before writing it down. If you exchange written material with another student (including email, discord chat, text messages, etc.), take notes while discussing with a fellow student, or copy from another person's screen, then this work is **not** yours.
  2. Cite all sources for material you hand in that is not your original work. Include these citations directly into the submitted PDF at the end of the question you used the source for. For example, if you reference a website when writing a proof, you should include a link to the website at the end of the question, along with a description as to how it was used. Use the complete URL so that the marker can check the source.
  3. Posting assignment questions to online forums (Chegg, Stack Overflow, etc.) is strictly prohibited. This is a copyright violation, and all cases will be immediately reported to the department.
  4. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. However, you may still get a low grade if you submit work that is not primarily your own. You will not be given credit for text or code that is clearly generated by another source.
  5. We will be looking for plagiarism in all submissions, possibly using automated software designed for the task.
  6. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize.

Questions:

1. Fred the frog is hopping along a row of n lilypad clusters. Each lilypad cluster consists of three distinct lilypads, which are coloured red, green, and blue. Each lilypad is also associated with a non-negative integer value, and if Fred lands on a lilypad he adds its value to his total score.

   Fred wants to accumulate the highest possible score by jumping across the row of clusters. He starts at the first cluster, and must land on each cluster in order. For each cluster, Fred can choose to land on the red, the green, or the blue lilypad, but he can only land on one lilypad per cluster and is not allowed to land on the **same colour twice in a row**.

   Your goal will be to write an algorithm that outputs the maximum total score Fred can earn by jumping across the lilypads. The problem specification is as follows:

   > **Input:** Three arrays of n integers each, denoted $r_1, r_2, \ldots r_n$, $g_1, g_2, \ldots g_n$, and $b_1, b_2, \ldots b_n$, where r_i, g_i and b_i represent the values of the red, green, and blue lilypads in the i^{th} cluster.
   > **Output:** The maximum total score that Fred can accumulate by jumping across the row of lilypads.

   As an example, on the input `r = [8,1,3,2]`, `g = [2,1,1,2]` and `b = [3,1,2,7]` the maximum score that Fred can accumulate is $8 + 1 + 3 + 7 = 19$.

   (a) (0.5 marks) Suppose you choose to solve this problem using the greedy heuristic "Pick the maximum-value lilypad from the next cluster that doesn't match the colour of Fred's current lilypad." Give a counterexample to show why this greedy heuristic doesn't give the optimal solution.

   (b) (0.5 marks) Now, suppose you switch to the heuristic "Consider the next **two** clusters. Out of the two lilypads on the next cluster that don't match the colour of Fred's current lilypad, pick the option that lets him make the maximum number of points over the next two jumps." Give a counterexample to show why this greedy heuristic also doesn't give the optimal solution.

   (c) (2 marks) Suppose that the lilypad weights are restricted to be either $+1$ or -1. Do either of the above greedy heuristics work in this case? Carefully explain why your answer is correct, either by proving the heuristic is correct or giving a counterexample.

   (d) (2 marks) Design a dynamic programming algorithm that solves this problem for the case where lilypad weights are allowed to be any integer. Give the recursive formulas, including all base cases, and describe the data structure you would use to handle the caching. Give the asymptotic runtime of your algorithm and justify your answer.

   Hint: you will probably need a "wrapper" function for this one, in addition to the recursive formula like we have been showing like in class, since you also need additional arguments to capture the fact that Fred cannot jump on the same colour twice. If you need to explain aspects of your algorithm in words, that is fine too.

2. (3 marks) **Longest Shared Vowel-Free Subsequence**: Suppose you are given two strings of letters a–z, which we call A and B, with lengths $|A| = n$ and $|B| = m$. Your goal is to find the **length** of the longest shared subsequence of letters from A and B, not counting vowels. For example, if

$$A = dynamicprogramming$$
$$B = dynamitemining,$$

the longest shared vowel-free subsequence would be $dynmmng$, which has length 7. Meanwhile, if

$$A = dynamicprogramming$$
$$B = rumor,$$

the longest shared vowel-free subsequence would be a tie between rr, mr, and rm, which all have length 2.

Design a dynamic programming algorithm that solves this problem. Give the recursive formulas, including all base cases, and describe the data structure you would use to handle the caching. Give the asymptotic runtime of your algorithm and justify your answer.

3. (2 marks) Recall the two-player coin game we discussed in class. Write a Python 3 function that, given an array of positive integer coin values and two integer arguments `start` and `end`, prints the maximum sum that can be won by the **current player** (the player presented with the input board), when the current line of coins is those that lie between indices `start` and `end` inclusive. Your script needs to return three values in a 3-tuple `(max, margin, takeRight)`:

   - `max` is the maximum total sum that can be won by the current player, even if the opponent plays optimally (as an integer)
   - `margin` is the margin by which the current player will win if both players play optimally (as an integer)
   - `takeRight` indicates which coin the current player should take for their next move to obtain this maximum (as a boolean, where `True` means to take the right coin and `False` means to take the left coin). If both moves yield the same score, then the your function should by default take the coin on the right.

   Your function should be contained in the Python 3 starter class provided on D2L. This class has a constructor with an argument named `coins`, which will be the array of coins you need to process. All values, including the inputs `start` and `end`, will be integers between 0 and 99,999. You **must** implement the dynamic programming algorithm presented in class to receive marks for this component.

   Upload `coin_game.py` to the correct location on Gradescope. Make sure your code is clean and well-commented.

Autograder notes:

Your code will be graded by an autograder on Gradescope. For your code to work with the autograder, you will need to use the template class `coin_game.py` provided on D2L. You should write your code to fill in the `run()` function **without changing the arguments**, and you must return (not print) a tuple containing the three values specified. You do not need to change the rest of the class. Do not change the filename, and do not upload any files other than your `coin_game.py`.

You can test your code with the autograder as many times as you like before submitting. It is also recommended that you add print statements for debugging during your own initial development. There is a set of test cases included in `a3test.py`, provided on D2L, that you can use to test your code locally if you prefer.

Note that marks will be allocated to your code structure and commenting, so pay attention to this too.

---

author: jcleahy@ucalgary.ca