CS 473 ✧ Fall 2022

Homework 2

Due Tuesday, September 13, 2022 at 9pm

---

1. (a) Recall that a *palindrome* is any string that is exactly the same as its reversal, like I or DEED or RACECAR or AMANAPLANACATACANALPANAMA. Describe and analyze an algorithm to find the length of the *longest subsequence* of a given string that is also a palindrome.

   For example, given the string MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM as input, your algorithm should return 11, which is the length of the longest palindrome subsequence MHYMRORMYHM.

   (b) Similarly, a *repeater* is any string whose first half and second half are identical, like MEME or MURMUR or HOTSHOTS or SHABUSHABU. Describe and analyze an algorithm to find the length of the *longest subsequence* of a given string that is also a repeater.

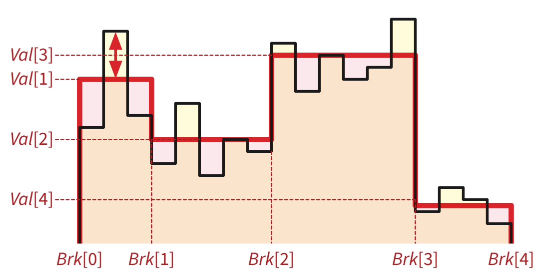
   For example, given the string AINTNOPARTYLIKEMYNANASTEAPARTYHEYHO as input, your algorithm should return 20, which is the length of the longest repeater subsequence ANTPARTYEYANTPARTYEY.

2. Describe and analyze an efficient algorithm to solve the following one-dimensional clustering problem. Given an unsorted array $Data[1..n]$ of real numbers, we want to cluster this data into k clusters, each represented by an interval of indices and a real value, so that the maximum error between any data point and its cluster value is minimized.

   More concretely, your algorithm should return an unsorted array $Val[1..k]$ of real numbers and a sorted array $Brk[0..k]$ of integer *breakpoints* such that $Brk[0] = 0$ and $Brk[k] = n$. For each index i, the ith interval covers items $Brk[i-1]+1$ through $Brk[i]$ in the input data, and the value of the ith interval is $Val[i]$. The output values $Val[i]$ are not necessarily elements of the input array. Your algorithm should compute arrays B and V that minimize the maximum absolute difference between any data point and the value of the unique interval that covers it:

$$Error(Data, Brk, Val) = \max\left\{|Data[i] - Val[j]| \;\middle|\; Brk[j-1] < i \leq Brk[j]\right\}$$

   We can visualize both the input data and the output approximation using bar charts, as shown in the figure below; the double arrow shows the error for this approximation.



Approximating 18 data points with four weighted intervals

3. You've been hired to store a sequence of n books on shelves in a library, using as little *vertical* space as possible. The *horizontal* order of the books is fixed by the cataloging system and cannot be changed; each shelf must store a contiguous interval of the given sequence of books. You can adjust the height of each shelf to match the tallest book on that shelf; in particular, you can change the height of any empty shelf to zero.

   You are given two arrays $H[1..n]$ and $W[1..n]$, where $H[i]$ and $W[i]$ are respectively the height and width of the ith book. Each shelf has the same fixed length L. Each book as width at most L, so every book fits on a shelf. The total width of all books on each shelf cannot exceed L. Your task is to shelve the books so that the *sum of the heights* of the shelves is as small as possible.

   (a) There is a natural greedy algorithm, which actually yields an optimal solution when all books have the same height: If $n > 0$, pack as many books as possible onto the first shelf, and then recursively shelve the remaining books.

      Show that this greedy algorithm does *not* yield an optimal solution if the books can have different heights. *[Hint: There is a small counterexample.]*

   (b) Describe and analyze an efficient algorithm to assign books to shelves to minimize the total height of the shelves.

Standard dynamic programming rubric. 10 points =

- 3 points for a clear and correct English description of the recursive function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.)
  - − 1 for naming the function "OPT" or "DP" or any single letter.
  - No credit if the description is inconsistent with the recurrence.
  - No credit if the description does not explicitly describe how the function value depends on the named input parameters.
  - No credit if the description refers to internal states of the eventual dynamic programming algorithm, like "the current index" or "the best score so far". The function must have a well-defined value that depends *only* on its input parameters (and constant global variables).
  - An English explanation of the *recurrence* or *algorithm* does not qualify. We want a description of *what* your function returns, not (here) an explanation of *how* that value is computed.
- 4 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 for base case(s). −½ for one *minor* bug, like a typo or an off-by-one error.
  - + 3 for recursive case(s). −1 for each *minor* bug, like a typo or an off-by-one error.
  - − 2 for greedy optimizations without proof, even if they are correct.
  - **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 3 points for iterative details
  - + 1 for describing (or sketching) an appropriate memoization data structure
  - + 1 for describing (or sketching) a correct evaluation order; a clear picture is usually sufficient. If you use nested for loops, be sure to specify the nesting order.
  - + 1 for correct time analysis. (It is not necessary to state a space bound.)
- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem specifically says otherwise.
- ***Iterative pseudocode is not required for full credit***. If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, memoization structure, or evaluation order. However, you ***do*** still need and English description of the underlying recursive function (or equivalently, the contents of the memoization structure). ***Perfectly correct iterative pseudocode, with no explanation or time analysis, is worth at most 6 points out of 10.***
- Partial credit for incomplete solutions depends on the running time of the ***best possible*** completion (up to the target running time). For example, consider a solution that contains *only* a clear English description of a function, with no recurrence or iterative details.
  - If the described function *can* be developed into an algorithm with the target running time, the solution is worth 3 points.
  - If the described function leads to an algorithm that is slower than the target time by a factor of n, the solution could be worth only 2 points (= 70% of 3, rounded).
  - If the described function cannot lead to a polynomial-time algorithm, it could be worth 1 or even 0 points.