CS 473 ✧ Fall 2022
Homework 3

Due Tuesday, September 20, 2022 at 9pm

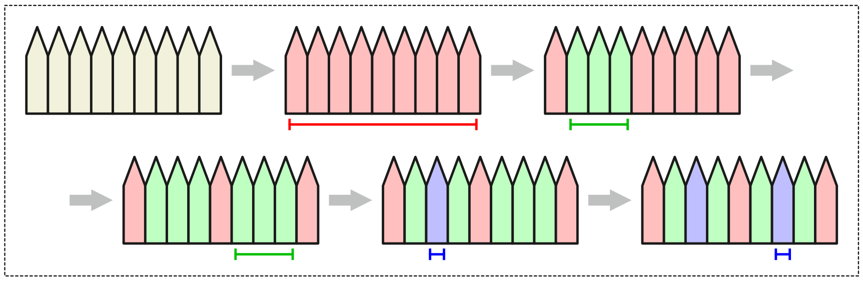
---

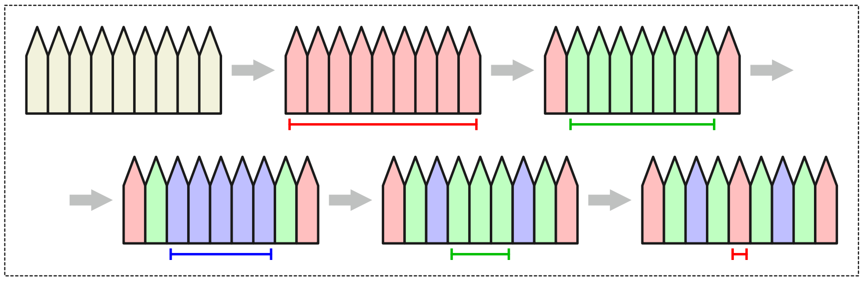
1. Huckleberry Sawyer needs to get his fence painted. His fence consists of a row of n wooden slats, all initially unpainted. He has been given a target color for each slat.

   Huck refuses to do any painting himself. Instead, each day he can hire one of his friends to paint any *contiguous* subset of slats a single color, for the uncomfortably high price of one nickel. Huck has really good paint, so he doesn't care if the same slat gets painted multiple times, as long as the last coat of paint on each slat matches its target color.

   Describe and analyze an algorithm that determines, given an array of target colors $C[1..n]$ as input, the minimum number of nickels that Huck must spend to have his fence completely painted.

   For example, if $n = 9$ and Huck's target color sequence is red, green, blue, green, red, green, blue, green, red, then your algorithm should return the integer 5. There are at least two different ways that Huck can get his fence painted by spending only five nickels:

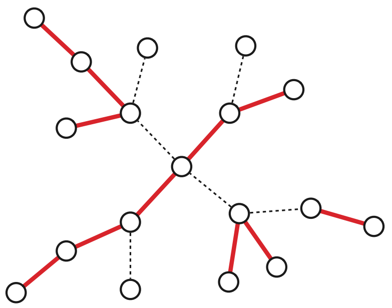




[Hint: You need to prove that there is always an optimal plan with a certain structure.]

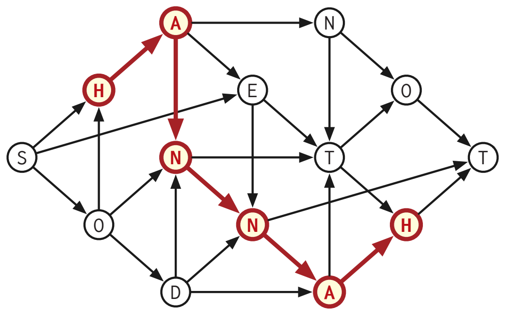
2. Let T be an arbitrary tree—a connected undirected graph with no cycles—each of whose edges has some positive weight. Describe and analyze an algorithm to cover the vertices of T with disjoint paths whose total length is as large as possible. (As usual, the length of a path is the sum of the weights of its edges.) Each vertex of T must lie on exactly one of the paths.

   The following figure shows a tree covered by seven disjoint paths, three of which have length zero.



3. Suppose we are given a directed acyclic graph G with labeled vertices. Every path in G has a label, which is a string obtained by concatenating the labels of its vertices in order. Recall that a *palindrome* is a string that is equal to its reversal.

   Describe and analyze an algorithm to find the length of the longest palindrome that is the label of a path in G. For example, given the graph below, your algorithm should return the integer 6, which is the length of the palindrome HANNAH.



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.