HW 6

Due on **Wednesday, October 14, 2020 at 10am CST**

CS/ECE 374: Algorithms & Models of Computation, Fall 2020 Version: **2.1**

Submission instructions as in previous homeworks.

16 (100 PTS.) Feline indiscretion advised.

Real cats plan out their day by drawing action cards from a deck. Each time they draw a card, they must either perform the action indicated by the card, or else discard it. The possible action cards are Nap, Yawn, Eat, Stretch, and Climb. There are a few rules governing the sequences of actions they perform:

- A Nap can only be followed by a Yawn or a Stretch.
- Eat can only follow a Climb.

Other than these rules, cats are free to choose any *subsequence* of action cards from the deck of cards they are given.

For example, given the following deck of cards

$$N, C, Y, S, C, E, E$$

The subsequence N, C, E would not be compatible with the rules (since Nap was followed by something other than Yawn or Stretch. One longest acceptable subsequence would be N, Y, S, C, E.

16.A. (50 PTS.) Given a deck of n action cards, give a backtracking algorithm for computing the length of the longest sequence of actions compatible with the rules above. Describe the asymptotic running time as a function of n.

16.B. (50 PTS.) Given a deck of n action cards, give a dynamic programming algorithm for computing the length of the longest sequence of actions compatible with the rules above. Describe the asymptotic running time as a function of n. The running time and space used by your algorithm should be as small as possible.

17 (100 PTS.) Cutting strings.

Assume you have an oracle that can query assess the "quality" of strings: $\mathbf{q} : \Sigma^* \rightarrow \mathbb{N}$. Computing $\mathbf{q}$ using the oracle takes $O(1)$ time regardless of the size of the string.

The following is an example $\mathbf{q}$ function (please note this is just an example, your algorithms must work for *any* $\mathbf{q}$ function):

$$\mathbf{q}(CATDOG) = 10, \qquad \mathbf{q}(CAT) = 9, \qquad \mathbf{q}(DOG) = 10, \qquad \mathbf{q}(ATDO) = 15,$$

where $\mathbf{q}$(everything else) $= 0$. For the given input string $x[1 \ldots n]$, you may invoke the $\mathbf{q}$ function by passing indexes. Thus, $\mathbf{q}(i, j)$ is a shorthand for $\mathbf{q}(x[i...j])$.

For each of the following, it is sufficient to describe how to compute the value realizing the desired optimal solution.

17.A. (20 PTS.) Give an algorithm to compute the highest quality substring of x.

For example, for the $\mathbf{q}$ function defined above, the highest quality substring of the string *CATDOG* is 15 (because the highest quality substring is *ATDO*). Analyze the performance of your algorithm. For full credit, you need to *prove* that your algorithm achieves the best possible asymptotic efficiency.

17.B. (40 PTS.) A *decomposition* of string x is a sequence of **non-empty** substrings $x_1, x_2, ..., x_k$ such that $x = x_1 x_2 ... x_k$. The *quality* of such a decomposition is

$$\mathbf{Q}(x_1, \ldots, x_k) = \min_i \mathbf{q}(x_i).$$

Namely, the quality is determined by the worst substring the decomposition uses.

Give a dynamic programming algorithm (and analyze its performance) to compute the decomposition that **maximizes** the quality $\mathbf{Q}$ of the decomposition (yeh, this is a bit confusing).

For example, if the string is *CATDOG*, then the best possible min-quality is 9. (*CAT*, *DOG* is a decomposition that achieves this).

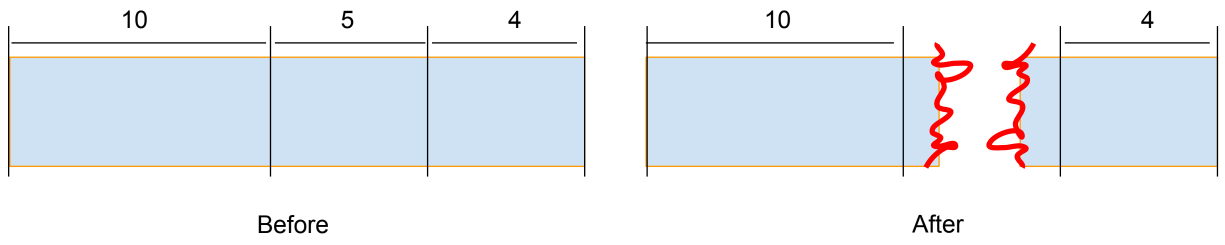
17.C. (40 PTS.) Give a dynamic programming algorithm (and analyze its performance) to find the best possible *average* quality of the **non-empty** substrings in a decomposition (that is, the sum quality of the substrings in a decomposition divided by the number of substrings in the decomposition).

For example, continuing the above examples, the best average quality decomposition for the string *CATDOG* is 10 (the decomposition *CATDOG* achieves this).

18 (100 PTS.) Laser Cuttery

You are in charge of programming the laser cutter at the local Maker Space. A member of the Maker Space needs your help to cut wood for a project. You're given a piece of wood to cut into smaller pieces. The wood is already marked according to positive integer-length segments, the lengths of which are given to you as a sequence.

Your laser cutter isn't tuned all that well, such that the only way to cut a piece of wood into two smaller pieces is to burn through one of the marked segments, destroying it. For example, the following illustrates the result of cutting a piece of wood marked as [10, 5, 4] into two smaller pieces [10] and [4].



The cost of each cut is proportional to the *total length* of the piece being cut. Thus the cut depicted above would cost \$19. (Assume after cutting into two smaller pieces, you can file the ends down for free).

It is also possible to cut off a segment at the end, so for example you could cut [5, 4] into [5], for a cost of \$9.

The project that hired you can only make use of pieces of wood length 5 or smaller. Anything larger is unusable. For example, pieces $[2, 3], [1], [5]$ would all be usable, but $[5, 4]$ would have to be thrown out or cut down further.

18.A. (40 PTS.) Suppose you need to produce usable wood pieces with a *total combined length* as large as possible. Cost is no object. Give an algorithm to compute the largest total combined length you could achieve. Analyze its asymptotic efficiency in terms of n, the number of marked segments in the initial piece of wood.

18.B. (40 PTS.) Suppose you get paid t dollars for each usable piece you produce, for some pre-specified t. Your net profits are the difference between the amount you get paid and the total cost of the cuts you make. Give an algorithm, as efficient as possible, to compute the best possible achievable net profit. Analyze its asymptotic efficiency in terms of n, the number of marked segments in the initial piece of wood.

18.C. (20 PTS.) Modify your algorithm for the previous part so that it outputs not just the lowest cost, but also the sequence of cuts necessary to achieve that cost.