CS 4349.003 Homework 4

Due Wednesday September 25th on eLearning

September 18, 2019

Please answer the following **2** questions (or a third for extra credit).

1. (a) Consider the following variant of the text segmentation problem. We are given a string $A[1..n]$ and a subroutine $IsWord$ that determines whether a given string is a word. For the variant, we are also given a non-negative integer w. We want to know whether A can be partitioned into a sequence of ***at most w words***.

   To solve this problem, let's define a function $SimplySplittable(i, k)$ that returns TRUE if and only if the suffix $A[i..n]$ can be partitioned into a sequence of at most k words. We need to compute $SimplySplittable(1, w)$.

   Observe that $SimplySplittable(i, k) =$ FALSE if $k < 0$, because you cannot partition a string into a negative number of words. On the other hand, $SimplySplittable(i, k) =$ TRUE if $i > n$ and $k \geq 0$, because the empty sequence of words contains $0 \leq k$ words. Finally, for $i \leq n$, a sequence of at most k words consists of a word followed by at most $k - 1$ additional words. Our function satisfies the recurrence

$$SimplySplittable(i,k) = \begin{cases} \text{FALSE} & \text{if } k < 0 \\ \text{TRUE} & \text{if } i > n \text{ and } k \geq 0 \\ \bigvee_{j=i}^{n} \left(\begin{array}{l} IsWord(i,j) \wedge \\ \quad SimplySplittable(j+1, k-1) \end{array} \right) & \text{otherwise} \end{cases}$$

   where $IsWord(i, j)$ is shorthand for $IsWord(A[i..j])$.

   Describe an iterative dynamic programming algorithm that determines if $A[1..n]$ can be partitioned into at most w words based using the above recurrence. You should following the steps for building solutions to the recurrence from the bottom up as described in Erickson 3.4. In particular, explain why your evaluation order is correct, and don't forget to analyze the running time of your algorithm. Your running time should be given in terms of n and w.

   (b) Now suppose are you given only the string $A[1..n]$ and the subroutine $IsWord$. Describe and analyze a dynamic programming algorithm that determines if $A[1..n]$ can be partitioned into an ***odd*** number of words. There are no other restrictions on how many words are used, just that the number of them is odd.

   For full credit, you ***must*** give a clear English description of the recursive subproblems you are solving, explain how to solve them recursively (using a recurrence), and then explain how to turn your recursive solution into an efficient dynamic programming algorithm. Again, see Erickson 3.4 for the full list of steps you should follow.

2. Let's define a *summary* of two strings A and B to be a concatenation of substrings of the following form:

   - ▲SNA indicates a substring SNA of only the first string A.
   - ♦FOO indicates a common substring FOO of both strings.
   - ▼BAR indicates a substring BAR of only the second string B.

   A summary is *valid* if we can recover the original strings A and B by concatenating the appropriate substrings of the summary in order and discarding the delimiters ▲, ♦, and ▼. Each regular character has length 1, and each delimiter ▲, ♦, or ▼ has some fixed non-negative length Δ. The *length* of a summary is the sum of the lengths of its symbols.

   For example, each of the following strings is a valid summary of the strings KITTEN and KNITTING:

   - ♦K▼N♦ITT▲E▼I♦N▼G has length $9 + 7\Delta$.
   - ♦K▼N♦ITT▲EN▼ING has length $10 + 5\Delta$.
   - ♦K▲ITTEN▼NITTING has length $13 + 3\Delta$.
   - ▲KITTEN▼KNITTING has length $14 + 2\Delta$.

   Describe and analyze an algorithm that computes the length of the shortest summary of two given strings $A[1..m]$ and $B[1..n]$. The delimiter length Δ is also part of the input to your algorithm. For example:

   - Given strings KITTEN and KNITTING and $\Delta = 0$, your algorithm should return 9 (for the first summary above).
   - Given strings KITTEN and KNITTING and $\Delta = 1$, your algorithm should return 15 (for the second summary above).
   - Given strings KITTEN and KNITTING and $\Delta = 2$, your algorithm should return 18 (for the forth summary above).

   A correct $O((m+n)mn)$ time algorithm is worth 8 points. A correct $O(mn)$ time algorithm is worth the full 10 points.

   [Hint: There is an $O((m+n)mn)$ time algorithm based on guessing the first special substring and recursively computing the rest of the summary; this solution is kind of like a combination of string segmentation and computing the edit distance. For $O(mn)$ time, how can you guess a single character of the summary and provide just enough information for the Recursion Fairy to handle the rest of the job?]

3. **(Extra credit)** Suppose you are given an $n \times n$ bitmap as an array $M[1..n, 1..n]$ of 0s and 1s. A *solid block* in M is a subarray of the form $M[i..i', j..j']$ in which all bits are equal.

   Describe and analyze an algorithm to find the maximum area of a solid block in M. An $O(n^3)$ time algorithm is worth half a homework problem of extra credit. A slower $O(n^4)$ time algorithm is worth a little less, and a faster $O(n^2 \log n)$ time algorithm is worth a little more.