CS/ECE 374 A ✧ Spring 2018

Homework 5

Due Tuesday, March 6, 2018 at 8pm

---

1. It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the list of n songs that the judges will play during the contest, in chronological order.

   You know all the songs, all the judges, and your own dancing ability extremely well. For each integer k, you know that if you dance to the kth song on the schedule, you will be awarded exactly $Score[k]$ points, but then you will be physically unable to dance for the next $Wait[k]$ songs (that is, you cannot dance to songs $k+1$ through $k+Wait[k]$). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

   Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays $Score[1..n]$ and $Wait[1..n]$.

2. Suppose you are given a NFA $M = (\{0,1\}, Q, s, A, \delta)$ and a binary string $w \in \{0,1\}^*$. Describe and analyze an efficient algorithm to determine whether M accepts w. Concretely, the input NFA M is represented as follows:

   - $Q = \{1, 2, \ldots, k\}$ for some integer k.
   - The start state s is state 1.
   - Accepting states are indicated by a boolean array $A[1..k]$, where $A[q] =$ TRUE if and only if $q \in A$
   - The transition function δ is represented by a boolean array $inDelta[1..k, 0..1, 1..k]$, where $inDelta[p, a, q] =$ TRUE if and only if $q \in \delta(p, a)$.

   Finally, the input string is given as an array $w[1..n]$. Your algorithm should return TRUE if M accepts w, and FALSE if M does not accept w. Report the running time of your algorithm as a function of k (the number of states in M) and n (the length of w). *[Hint: Do not convert M to a DFA!!]*

3. Recall that a ***palindrome*** is any string that is exactly the same as its reversal, like the empty srting, or `I`, or `DEED`, or `RACECAR`, or `AMANAPLANACATACANALPANAMA`.

   Any string can be decomposed into a sequence of palindromes. For example, the string `BUBBASEESABANANA` ("Bubba sees a banana.") can be broken into non-empty palindromes in the following ways (and 65 others):

   `BUB` • `BASEESAB` • `ANANA`

   `B` • `U` • `BB` • `ASEESA` • `B` • `ANANA`

   `BUB` • `B` • `A` • `SEES` • `ABA` • `N` • `ANA`

   `B` • `U` • `BB` • `A` • `S` • `EE` • `S` • `A` • `B` • `A` • `NAN` • `A`

   `B` • `U` • `B` • `B` • `A` • `S` • `E` • `E` • `S` • `A` • `B` • `A` • `N` • `A` • `N` • `A`

   (a) Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example:

   - Given the string `PALINDROME`, your algorithm should return the integer 10.
   - Given the string `BUBBASEESABANANA`, your algorithm should return the integer 3.
   - Given the string `RACECAR`, your algorithm should return the integer 1.

   (b) A ***metapalindrome*** is a decomposition of a string into a sequence of non-empty palindromes, such that the sequence of palindrome lengths is itself a palindrome. For example, the decomposition

   `BUB` • `B` • `ALA` • `SEES` • `ABA` • `N` • `ANA`

   is a metapalindrome for the string `BUBBALASEESABANANA`, with the palindromic length sequence $(3, 1, 3, 4, 3, 1, 3)$. Describe and analyze an efficient algorithm to find the length of the shortest metapalindrome for a given string. For example:

   - Given the string `BUBBALASEESABANANA`, your algorithm should return the integer 7.
   - Given the string `PALINDROME`, your algorithm should return the integer 10.
   - Given the string `DEPOPED`, your algorithm should return the integer 1.

Solved Problem

4. A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, the string BANANAANANAS is a shuffle of the strings BANANA and ANANAS in several different ways.

BANANA$_{\text{ANANAS}}$ BAN$_{\text{ANA}}$ANA$_{\text{NAS}}$ B$_{\text{AN}}$AN$_{\text{A}}$A$_{\text{NA}}$NA$_{\text{S}}$

Similarly, the strings PRODGYRNAMAMMIINCG and DYPRONGARMAMMICING are both shuffles of DYNAMIC and PROGRAMMING:

PRO$^{\text{D}}$G$^{\text{Y}}$R$^{\text{NAM}}$AMMI$^{\text{I}}$N$^{\text{C}}$G DY$_{\text{PRO}}$N$_{\text{G}}$A$_{\text{R}}$M$_{\text{AMM}}$IC$_{\text{ING}}$

Given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, describe and analyze an algorithm to determine whether C is a shuffle of A and B.

Solution: We define a boolean function $Shuf(i, j)$, which is TRUE if and only if the prefix $C[1..i+j]$ is a shuffle of the prefixes $A[1..i]$ and $B[1..j]$. This function satisfies the following recurrence:

$$Shuf(i,j) = \begin{cases} \text{TRUE} & \text{if } i = j = 0 \\ Shuf(0, j-1) \wedge (B[j] = C[j]) & \text{if } i = 0 \text{ and } j > 0 \\ Shuf(i-1, 0) \wedge (A[i] = C[i]) & \text{if } i > 0 \text{ and } j = 0 \\ \big(Shuf(i-1, j) \wedge (A[i] = C[i+j])\big) & \\ \quad \vee \big(Shuf(i, j-1) \wedge (B[j] = C[i+j])\big) & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

We need to compute $Shuf(m, n)$.

We can memoize all function values into a two-dimensional array $Shuf[0..m][0..n]$. Each array entry $Shuf[i, j]$ depends only on the entries immediately below and immediately to the right: $Shuf[i-1, j]$ and $Shuf[i, j-1]$. Thus, we can fill the array in standard row-major order. The original recurrence gives us the following pseudocode:

```
SHUFFLE?(A[1..m], B[1..n], C[1..m+n]):
  Shuf[0,0] ← TRUE
  for j ← 1 to n
      Shuf[0,j] ← Shuf[0,j-1] ∧ (B[j] = C[j])
  for i ← 1 to n
      Shuf[i,0] ← Shuf[i-1,0] ∧ (A[i] = B[i])
      for j ← 1 to n
          Shuf[i,j] ← FALSE
          if A[i] = C[i+j]
              Shuf[i,j] ← Shuf[i,j] ∨ Shuf[i-1,j]
          if B[i] = C[i+j]
              Shuf[i,j] ← Shuf[i,j] ∨ Shuf[i,j-1]
  return Shuf[m,n]
```

The algorithm runs in $O(mn)$ *time*. ■

Rubric: Max 10 points: Standard dynamic programming rubric. No proofs required. Max 7 points for a slower polynomial-time algorithm; scale partial credit accordingly.

Standard dynamic programming rubric. For problems worth 10 poins:

- 6 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 point for a clear English description of the function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.) **Deadly Sin: Automatic zero if the English description is missing.**
  - + 1 point for stating how to call your function to get the final answer.
  - + 1 point for base case(s). −½ for one *minor* bug, like a typo or an off-by-one error.
  - + 3 points for recursive case(s). −1 for each *minor* bug, like a typo or an off-by-one error. **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 4 points for details of the dynamic programming algorithm
  - + 1 point for describing the memoization data structure
  - + 2 points for describing a correct evaluation order; a clear picture is usually sufficient. If you use nested loops, be sure to specify the nesting order.
  - + 1 point for 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 says otherwise.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, ***but 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. (But you still need to describe the underlying recursive function in English.)
- Official solutions will provide target time bounds. Algorithms that are faster than this target are worth more points; slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of n. Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.

  We rarely include these target time bounds in the actual questions, because when we have included them, significantly more students turned in algorithms that meet the target time bound but didn't work (earning 0/10) instead of correct algorithms that are slower than the target time bound (earning 8/10).