CS/ECE 374 A ✧ Fall 2023

Homework 6

Due Tuesday, October 10, 2023 at 9pm Central Time

Please make sure that you read and understand the standard dynamic programming rubric.

---

1. Satya is in charge of establishing a new testing center for the Standardized Awesomeness Test (SAT), and found an old conference hall that is perfect. The conference hall has n rooms of various sizes along a single long hallway, numbered in order from 1 through n. Satya knows exactly how many students fit into each room, and he wants to use a subset of the rooms to host as many students as possible for testing.

   Unfortunately, there have been several incidents of students cheating at other testing centers by tapping secret codes through walls. To prevent this type of cheating, Satya can use two adjacent rooms only if he demolishes the wall between them. The city's chief architect has determined that demolishing the walls on both sides of the same room would threaten the building's structural integrity. For this reason, Satya can never host students in three consecutive rooms.

   Describe an efficient algorithm that computes the largest number of students that Satya can host for testing without using three consecutive rooms. The input to your algorithm is an array $S[1..n]$, where each $S[i]$ is the (non-negative integer) number of students that can fit in room i.

2. As a typical overworked college student, you occasionally pull all-nighters to get more work done. Painful experience has taught you that the longer you stay awake, the less productive you are.

   Suppose there are n days left in the semester. For each of the next n days, you can either stay awake and work, or you can sleep. You have an array $Score[1..n]$, where $Score[i]$ is the (always positive) number of points you will earn on day i *if you are awake and well-rested*.

   However, staying awake for several days in a row has a price: Each consecutive day you stay awake cuts the quality of your work in half. Thus, if you are awake on day i, and you most recently slept on day $i-k$, then you will actually earn $Score[i]/2^{k-1}$ points on day i. (You've already decided to sleep on day 0.)

   For example, suppose $n = 6$ and $Score = [3, 7, 4, 3, 9, 1]$.

   - If you work on all six days, you will earn $3 + \frac{7}{2} + \frac{4}{4} + \frac{3}{8} + \frac{9}{16} + \frac{1}{32} = 8.46875$ points.
   - If you work only on days 1, 3, and 5, you will earn $3 + 4 + 9 = 16$ points.
   - If you work only on days 2, 3, 5, and 6, you will earn $7 + \frac{4}{2} + 9 + \frac{1}{2} = 18.5$ points.

   Design and analyze an algorithm that computes the maximum number of points you can earn, given the array $Score[1..n]$ as input. For example, given the input array $[3, 7, 4, 3, 9, 1]$, your algorithm should return the number 18.5.

   VERY IMPORTANT: Do not actually do this in real life!

3. ***Practice only. Do not submit solutions.***

(a) Any string can be decomposed into a sequence of palindromes. For example, the string BUBBASEESABANANA ("Bubba sees a banana.") can be broken into 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

Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string BUBBASEESABANANA, your algorithm should return 3.

(b) A ***metapalindrome*** is a decomposition of a string into a sequence of palindromes, such that the sequence of palindrome lengths is itself a palindrome. For example, the string BOBSMAMASEESAUKULELE ("Bob's mama sees a ukulele") has the following metapalindromes (among others):

BOB • S • MAM • ASEESA • UKU • L • ELE

B • O • B • S • M • A • M • A • S • E • E • S • A • U • K • U • L • E • L • E

The length sequences of these metapalindromes are $(3, 1, 3, 6, 3, 1, 3)$ and $(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$; notice that both of these sequences are themselves palindromes.

Describe and analyze an efficient algorithm to find the smallest number of palindromes in any metapalindrome for a given string. For example, given the input string BOBSMAMASEESAUKULELE, your algorithm should return 7.

Solved Problems

3. 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.

BANANAANANAS **BAN**ANA**ANA**NAS B**A**N**AN**A**A**NA**NA**S

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

PRO**D**G**Y**R**NAM**AMMI**I**N**C**G **DY**PRO**N**G**A**R**M**AMM**IC**ING

(a) 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 $B[1..j]$. We need to compute $Shuf(m,n)$. The function $Shuf$ satisfies the following recurrence:

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

We can memoize this function into a two-dimensional array $Shuf[0..m][0..n]$. Each array entry $Shuf[i,j]$ depends only on the entries immediately above and immediately to the left: $Shuf[i-1,j]$ and $Shuf[i,j-1]$. Thus, we can fill the array in standard row-major order in ***O(mn) time***. ■

Solution: The following algorithm runs in ***O(mn) time***.

```
IsShuffle?(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−1,j]
      if B[i] = C[i+j]
          Shuf[i,j] ← Shuf[i,j] ∨ Shuf[i,j−1]
  return Shuf[m,n]
```

Here $Shuf(i,j) = \text{TRUE}$ if and only if the prefix $C[1..i+j]$ is a shuffle of the

prefixes $A[1..i]$ and $B[1..j]$. ■

Rubric: 5 points, standard dynamic programming rubric. **Each of these solutions is separately worth full credit.** These are not the only correct solutions. −½ for reporting running time as $O(n^2)$. 3 points for a slower polynomial-time algorithm; scale partial credit accordingly.

(b) Given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, describe and analyze an algorithm to determine *the number of different ways* that A and B can be shuffled to obtain C.

Solution: Let $\#Shuf(i,j)$ denote the number of different ways that the prefixes $A[1..i]$ and $B[1..j]$ can be shuffled to obtain the prefix $C[1..i+j]$. We need to compute $\#Shuf(m,n)$.

The $\#Shuf$ function satisfies the following recurrence. Here I am using Iverson bracket notation to convert booleans to integers: For any proposition P, the expression $[P]$ is equal to 1 if P is true and 0 if P is false.

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

We can memoize this function into a two-dimensional array $\#Shuf[0..m][0..n]$. As in part (a), we can fill this array in standard row-major order in ***O(mn) time***. ■

Solution: The following algorithm runs in ***O(mn) time***:

```
NumShuffles(A[1..m], B[1..n], C[1..m+n]):
  #Shuf[0,0] ← 1
  for j ← 1 to n
      #Shuf[0,j] ← 0
      if (B[j] = C[j])
          #Shuf[0,j] ← #Shuf[0,j−1]
  for i ← 1 to n
      #Shuf[0,j] ← 0
      if (A[i] = B[i])
          #Shuf[0,j] ← #Shuf[i−1,0]
  for j ← 1 to n
      #Shuf[i,j] ← 0
      if A[i] = C[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]
```

Here $\#Shuf[i, j]$ stores the number of different ways that the prefixes $A[1..i]$ and $B[1..j]$ can be shuffled to obtain the prefix $C[1..i+j]$. ■

Rubric: 5 points, standard dynamic programming rubric. **Again, each of these solutions is separately worth full credit.** These are not the only correct solutions. −½ for reporting running time as $O(n^2)$. 3 points for a slower polynomial-time algorithm; scale partial credit accordingly.