CS 8803 GA: HW 1: Dynamic Programming

1. Palindrome substring.

A substring is *palindromic* if it is the same whether read left to right or right to left. For instance, the sequence

$$A, C, G, C, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic substrings, including: any single character; { C, T, G, T, C}; and { A, A, A, A }.

Devise a dynamic programming algorithm that takes a string $X = \{x_1, x_2, \ldots, x_n\}$ and returns the (length of the) longest palindromic substring.

(Faster (and correct) algorithm in $O(\cdot)$ notation is worth more credit.)

(1a) Define the entries of your table in words. E.g., $T(i)$ is ..., or $T(i,j)$ is ....

$T(i,j) = 1$ if substring $x_i, \ldots, x_j$ is a palindrome, else 0

(1b) State the recurrence for the entries of your table in terms of smaller subproblems.

$$T(i,j) = \begin{cases} 0 : & \text{if } i > j \\ 1 & \text{if } i = j \\ 1 & \text{if } j - i = 1 \text{ and } x_i = x_j \\ T(i+1, j-1) & \text{if } x_i = x_j \\ 0 & \text{otherwise} \end{cases}$$

(1c) Write pseudocode for your algorithm to solve this problem.

$L = 1$

for $i = 1 \rightarrow N$:
  $T(i,i) = 1$
for $i = 1 \rightarrow N-1$:
  if $x_i = x_{i+1}$:
    $T(i,i+1) = 1$
    $L = 2$
for $k = 2 \rightarrow N$:
  for $i = 1 \rightarrow N-k$:
    $j = i+k$
    if $x_i = x_j$ and $T(i+1, j-1) = 1$:
      $T(i,j) = 1$
      $L = j-i+1$
Return L

(1d) Analyze the running time of your algorithm.

2 loops over n + one nested loop of $n \times n/2$

for $O(n^2)$

2. Maximum Product.

The input to the problem is a string $Z = z_1 z_2 \ldots z_n$ where each $z_i \in \{1, 2, \ldots, 9\}$ and an integer k where $0 \leq k < n$. An example string is $Z = 8473817$, which is of length $n = 7$. We want to insert k multiplication operators $\times$ into the string so that the mathematical result of the expression is the largest possible. There are $n - 1$ possible locations for the operators, namely, after the i-th character where $i = 1, \ldots, n - 1$. For example, for input $Z = 21322$ and $k = 2$, then one possible way to insert the $\times$ operators is: $2 \times 1 \times 322 = 644$, another possibility is $21 \times 3 \times 22 = 1386$.

Design a dynamic programming to **output the maximum product** obtainable from inserting exactly k multiplication operators $\times$ into the string. You can assume that all the multiplication operations in your algorithm take O(1) time.

(Faster (and correct) algorithm in $O(\cdot)$ notation is worth more credit.)

(2a) Define the entries of your table in words. E.g., $T(i)$ is ..., or $T(i, j)$ is ....

$T(i,j)$ = max product using the first i digits, and using exactly j multiplication operations.

(2b) State the recurrence for the entries of your table in terms of smaller subproblems.

$$T(i,j) = \max_{1 \leq a < i} \left(T(a, j-1) \cdot x_{a+1} \ldots x_i \right)$$

(2c) Write pseudocode for your algorithm to solve this problem.

for $i = 1 \to n$
  $T(i,0) = x_1 \dots x_i$
  for $j = 1 \to k$
    $L = 0$
    for $a = 1 \to i-1$
      $L = \max(L, T(a, j-1) \cdot x_{a+1} \dots x_i)$
Return L

(2d) Analyze the running time of your algorithm.

2 nested loop of $O(n)$ for each $k \Rightarrow$
$O(kn^2)$

3. Coin changing variant.

This is a different variant of the coin changing problem. You are given denominations $x_1, x_2, \dots, x_n$ and you want to make change for a value B. You can **use each denomination at most once** and you can use at most k coins.

Input: Positive integers $x_1, \dots, x_n, B, k$.

Output: True/False whether or not there is a subset of coins with value B where each denomination is used at most once and at most k coins are used.

Design a dynamic programming algorithm for this problem.

(Faster (and correct) algorithm in $O(\cdot)$ notation is worth more credit.)

(3a) Define the entries of your table in words. E.g., $T(i)$ is ..., or $T(i,j)$ is ....

$T(i,j)$ = min number of coins needed to sum to j using the first i coins

(3b) State the recurrence for the entries of your table in terms of smaller subproblems.

$$T(i,j) = \begin{cases} \min(T(i-1,j), T(i-1, j-x_i)+1 & \text{if } x_i \le j \\ T(i-1,j) & \text{otherwise} \end{cases}$$

(3c) Write pseudocode for your algorithm to solve this problem.

$T(0,0) = 0$

for $j = 1 \to B$

  $T(0,j) = \infty$

for $i = 1 \to n$

  for $j = 1 \to B$

    if $x_i > j$

      $T(i,j) = T(i-1,j)$

    else

      $T(i,j) = \min(T(i-1,j), T(i-1, j-x_i)+1)$

Return $T(N,B) \leq K$

(3d) Analyze the running time of your algorithm.

$O(B) + O(n) \cdot O(B \Rightarrow O(nB)$