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 ....

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

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

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

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 $0 \le k < n$. 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 ....

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

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

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

3. Coin changing variant.

This is a different variant of the coin changing problem. You are given denominations $x_1, x_2, \ldots, x_n$ 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 $x_1, \ldots, x_n, B, k$.

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 ....

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

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

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