Problem 1: Longest Common Sub*!?*

Given two strings $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_m$ **give a dynamic programming algorithm** to find the **length** k of the longest string $Z = z_1, \ldots, z_k$ where Z appears as a **substring** of X and as a **subsequence** of Y. Recall, a substring is **consecutive** elements.

For example, for the following input:

$$\begin{aligned} X &= a, \mathbf{b}, \mathbf{d}, \mathbf{b}, \mathbf{a}, b, f, g, d \\ Y &= \mathbf{b}, e, t, f, \mathbf{d}, \mathbf{b}, f, \mathbf{a}, f, r \end{aligned}$$

then the answer is 4 (since, b, d, b, a is a substring of X and it is also a subsequence of Y). You do not need to output the actual substring, just its length.

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

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

Solution.

$T(i, j)$ is the length of the longest substring of $x_1 x_2 \ldots x_i$ **and includes** x_i that is also a subsequence of $y_1 y_2 \ldots y_j$.

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

Solution.

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

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

Solution.

```
for i := 1 to n set T(i,0) := 0
for j := 1 to m set T(0,j) := 0
for i := 1 to n do
    for j := 1 to m do
        if x_i = y_j then
            T(i,j) := T(i-1, j-1) + 1
        else
            T(i,j) := T(i, j-1)
return(max_{1<=i<=n} T(i,m))
```

(d) Analyze the running time of your algorithm.

Solution.

Each entry in $T(i, j)$ takes constant time to compute, since each is one of two expressions. There are nm entries in T, so this algorithm takes $O(nm)$ time.

Problem 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.)

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

Solution.

$T(i, j)$ is the maximum product obtainable from inserting exactly j multiplication operators $\times$ into the prefix $z_1 z_2 \ldots z_i$, for $0 \leq j < i \leq n$.
The answer to the original problem is $T(n, k)$.

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

Solution.

$$T(i, j) = \max_{\ell}\{T(\ell, j-1) \times (z_{\ell+1} \ldots z_i) : j \leq \ell < i\}$$

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

Solution.

```
for i = 1 → n
    set T(i,0) = z_1 ... z_i

for j = 1 → k
    for i = j+1 → n
        set T(i,j) = max_{j≤ℓ<i}{T(ℓ, j−1) × (z_{ℓ+1} ... z_i)}

return(T(n,k))
```

(d) Analyze the running time of your algorithm.

Solution.

Each entry in $T(i, j)$ takes $O(n)$ time to compute, since each is the maximum over up to n expressions. There are nk entries in T, so this algorithm takes $O(n^2k)$ time.

Problem 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.)

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

Solution.

$$T(i,b) = \begin{cases} \text{minimum number of coins with denominations } \{x_1, \ldots, x_i\}, \text{ whose values} \\ \quad \text{add up to exactly } b, \text{ where each denomination is used at most once} \\ \infty, \text{ if there is no combination of coins with denominations } \{x_1, \ldots, x_i\} \\ \quad \text{whose values add up to exactly } b \end{cases}$$

For the answer, if $T(n, B) \leq k$ then return(TRUE), while if $T(n, B) > k$ then return(FALSE).

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

Solution.

$$T(i,b) = \begin{cases} \min\{T(i-1,b), T(i-1,b-x_i)+1\} & x_i \leq b \\ T(i-1,b) & x_i > b \end{cases}$$

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

Solution.

```
for i := 0 to n set T(i,0) := 0
for b := 1 to B set T(0,b) := ∞
for i := 1 to n do
    for b := 1 to B do
        if x_i ≤ b then
            T(i,b) := min{T(i-1,b), T(i-1,b-x_i)+1}
        else
            T(i,b) := T(i-1,b)
if T(n,B) ≤ k then
    return(TRUE)
else
    return(FALSE)
```

(d) Analyze the running time of your algorithm.

Solution.

Each entry in $T(i, j)$ takes constant time to compute, since each is one of two expressions. There are nB entries in T, so this algorithm takes $O(nB)$ time.