2023-11-04 - Handout – Patterns of DP

Linear Dynamic Programming:

198. House Robber

Easy 3852 119 Add to List Share

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

Example 1:

```
Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
             Total amount you can rob = 1 + 3 = 4.
```

Example 2:

```
Input: [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house
```

Grid Dynamic Programming

There is a robot on an `m x n` grid. The robot is initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

Given the two integers `m` and `n`, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to `2 * 10^9`.

Example 1:


```
Input: m = 3, n = 7
Output: 28
```

Example 2:

```
Input: m = 3, n = 2
Output: 3
Explanation: From the top-left corner, there are a total of 3 ways to reach the bottom-right
corner:
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```

Caption

0/1 Knapsack

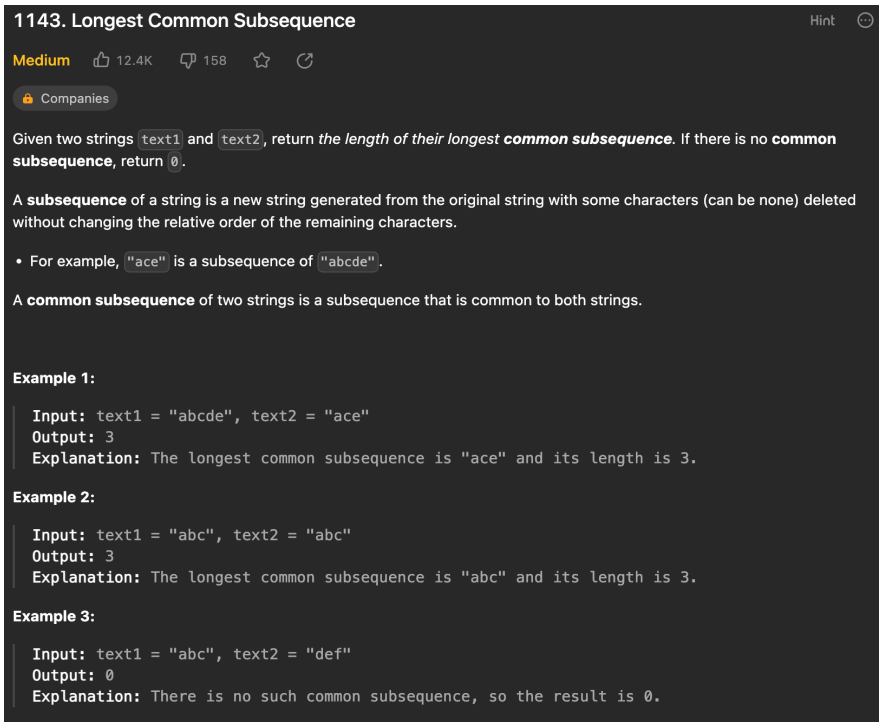
You're a burglar with a knapsack that can hold a total weight of `capacity`. You have a set of items (`n` items) each with fixed weight capacities and values. The weight and value are represented in an integer array. Create a function `knapsack()` that finds a subset or number of these items that will maximize value but whose total weight does not exceed the given number `capacity`.

Caption

Knapsack Unbounded

Given the weights and values of 'N' items, put these items in a knapsack of capacity 'W' to get the maximum total value in the knapsack with repetitions of items allowed.

```
Input:
W = 100
val = [10, 30, 20]
wt = [5, 10, 15]
Output:
300
```



1143. Longest Common Subsequence

Hint

Medium 12.4K 158

Companies

Given two strings `text1` and `text2`, return *the length of their longest* ***common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

Example 2:

```
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its length is 3.
```

Example 3:

```
Input: text1 = "abc", text2 = "def"
Output: 0
Explanation: There is no such common subsequence, so the result is 0.
```

Caption