TIP102 | Intermediate Technical Interview Prep

Intermediate Technical Interview Prep Spring 2025 (@ Section 3 | Tuesdays and Thursdays 6PM - 8PM PT)
Personal Member ID#: 117667

Session 1: Dynamic Programming

Session Overview

In this session, students will learn how to use a dynamic programming approach and memoization to optimize solutions by breaking down problems into overlapping subproblems and storing results to avoid redundant calculations. Dynamic programming is a key technique for improving the efficiency of algorithms, especially in optimization problems that would otherwise have high time complexity due to repeated work.

> You can find all resources from today including session slide decks, session recordings, and more on the resources tab

🎢 Part 1 : Instructor Led Session

We'll spend the first portion of the synchronous class time in large groups, where the instructor will lead class instruction for 30-45 minutes.

👨‍💻 Part 2: Breakout Session

In breakout sessions, we will explore and collaboratively solve problem sets in small groups. Here, the **collaboration, conversation, and approach** are just as important as “solving the problem” - please engage warmly, clearly, and plentifully in the process!

In breakout rooms you will:

- Screen-share the problem/s, and verbally review them together
- Screen-share an interactive coding environment, and talk through the steps of a solution approach
    - ProTip: - An Integrated Development Environment (IDE) is a fancy name for a tool you could use for shared writing of code - like Replit.com, Collabed.it, CodePen.io, or other - your staff team will specify which tool to use for this class!
- Screen-share an implementation of your proposed solution

- Independently follow-along, or create an implementation, in your own IDE.

Your program leader/s will indicate which code sharing tool/s to use as a group, and will help break down or provide specific scaffolding with the main concepts above.

▶ **Note on Expectations**

🔎 Problem Solving Approach

To build a long-term organized approach to problem solving, we'll start with three main steps. We'll refer to them as **UPI: Understand, Plan, and Implement**.

We'll apply these three steps to most of the problems we'll see in the first half of the course.

We will learn to:

- **Understand** the problem,
- **Plan** a solution step-by-step, and
- **Implement** the solution

▶ **Comment on UPI**

▶ **UPI Example**

Breakout Problems Session 1

▶ Standard Problem Set Version 1

▶ Standard Problem Set Version 2

▼ Advanced Problem Set Version 1

Problem 1: Spirit World Wisdom Pyramid

In *Avatar: The Last Airbender* the Spirit World is a parallel plane of existence where spirits live. In the Spirit World, there exists an ancient pyramid known as the Wisdom Pyramid. Each layer of the pyramid contains mystical numbers that guide the Avatar's journey. The numbers in each layer are formed by combining the two numbers directly above them from the previous layer.

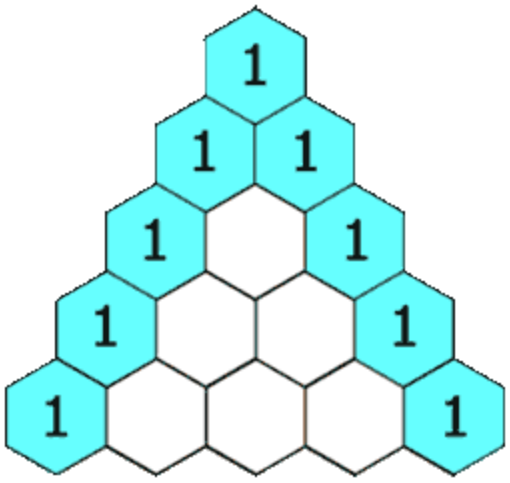
Given an integer `wisdomLevel`, return the `wisdomLevel`-th (0-indexed) row of the Wisdom Pyramid.

In the Wisdom Pyramid, each number is the sum of the two numbers directly above it from the previous row. The first row of the pyramid (layer 0) starts with the number 1.

Write a function `wisdom_pyramid()` to return the `wisdomLevel`-th row of the pyramid.

```
def wisdom_pyramid(wisdomLevel):
    pass
```

Example Usage:




```
print(wisdom_pyramid(3))
print(wisdom_pyramid(0))
print(wisdom_pyramid(5))
```

Example Output:

```
[1, 3, 3, 1]
Example 1 Explanation: The 3rd row of the Wisdom Pyramid is [1, 3, 3, 1].

[1]
Example 2 Explanation: The 0th row of the Wisdom Pyramid is [1].

[1, 5, 10, 10, 5, 1]
Example 3 Explanation: The 5th row of the Wisdom Pyramid is [1, 5, 10, 10, 5, 1].
```

▶ 💡 **Hint: Dynamic Programming**

Problem 2: Aang and Zuko's Elemental Duel

Aang and Zuko take turns in a strategic duel to control the elements, with Aang going first. The duel begins with a number `n` representing the strength of the elements on the battlefield.

On each turn, a player must:

1. Choose any `x` such that `0 < x < n` and `n % x == 0`.
2. Reduce the battlefield's strength by `x`, replacing `n` with `n - x`.

If a player cannot make a move, they lose the duel.

Write a function `aang_wins()` that accepts an integer `n` and returns `True` if Aang wins the duel (assuming both Aang and Zuko play optimally), and `False` otherwise.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

```
def aang_wins(n):
    pass
```

Example Usage:

```
print(aang_wins(2))
print(aang_wins(3))
```

Example Output:

```
True
Example 1 Explanation: Aang reduces the strength by 1, and Zuko has no more moves.

False
Example 2 Explanation: Aang reduces the strength by 1, then Zuko does the same, leav
Aang with no moves.
```

Problem 3: Iroh's Tea Brewing

Uncle Iroh is experimenting with various ingredients to brew the strongest tea. He has a list of ingredients, each with a specific strength value, represented by an integer array `ingredients`. Some ingredients increase the tea's strength (positive values), while others weaken it (negative values). Iroh needs to find the combination of consecutive ingredients that results in the tea with the highest strength value.

Write a function `strongest_tea()` to help Uncle Iroh find the subarray of consecutive ingredients that results in the largest product of strengths, and return that product.

```
def strongest_tea(ingredients):
    pass
```

Example Usage:

```
print(strongest_tea([1, 2, -3, 4]))
print(strongest_tea([-2, -1]))
```

Example Output:

```
4
Example 1 Explanation: The strongest tea Iroh can brew uses only the last ingredient

2
Example 2 Explanation: The strongest tea Iroh can brew uses both ingredients: -2 and
```

Problem 4: Toph and Katara's Training Synchronization

Toph and Katara are practicing their bending skills together. Both have different training sequences, but they want to synchronize their moves as much as possible. A synchronized sequence is a common subsequence that appears in both Toph's and Katara's training routines while preserving the order of the moves.

Given two strings `katara_moves` and `toph_moves`, 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 by deleting some characters (without changing the relative order of the remaining characters).

Using dynamic programming, write a function `training_synchronization()` to calculate the longest common subsequence.

```
def training_synchronization(katara_moves, toph_moves):
    pass
```

Example Usage:

```
print(training_synchronization("waterbend", "earthbend"))
print(training_synchronization("bend", "bend"))
print(training_synchronization("fire", "air"))
```

Example Output:

```
5
Example 1 Explanation: The longest common subsequence is "bend" and its length is 5.

4
Example 2 Explanation: The longest common subsequence is "bend" and its length is 4.

0
Example 3 Explanation: There is no common subsequence between "fire" and "air", so t
```

▶ 💡 **Hint: 2-D Dynamic Programming**

Problem 5: Aang's Airbending Journey

Aang is on a journey across a series of floating air platforms, represented by an integer array `platforms`. He starts on the first platform (index `0`), and each element in the array represents the maximum distance Aang can airbend-jump from that platform.

Write a function `aang_journey()` that returns `True` if Aang can reach the last platform, or `False` if it's impossible.

```
def aang_journey(platforms):
    pass
```

Example Usage:

```
print(aang_journey([2, 3, 1, 1, 4]))
print(aang_journey([3, 2, 1, 0, 4]))
```

Example Output:

```
True
Example 1 Explanation: Aang can airbend-jump 1 step from platform 0 to platform 1, a

False
Example 2 Explanation: No matter what, Aang is stuck at platform 3 because its maxim
```

Problem 6: Katara's Waterbending Mastery

Katara is working on perfecting her waterbending techniques, starting from a basic form `form1` and needing to adapt it to a more advanced form `form2`. To master the advanced form, Katara can perform three types of operations:

- **Insert** a move
- **Delete** a move
- **Replace** a move with another

Write a function `waterbending_mastery()` that calculates the minimum number of operations Katara needs to convert `form1` to `form2`.

```
def waterbending_mastery(form1, form2):
    pass
```

Example Usage:

```
print(waterbending_mastery("tide", "wave"))
print(waterbending_mastery("intention", "execution"))
```

Example Output:

```
3
Example 1 Explanation:
tide -> wide (replace 't' with 'w')
wide -> wade (replace 'i' with 'a')
wade -> wave (replace 'd' with 'v')

5
Example 2 Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```

Close Section

▼ Advanced Problem Set Version 2

Problem 1: Counting Pikachu's Thunderbolt Charges

Pikachu is preparing for an epic battle, and Ash needs to keep track of how many Thunderbolt charges Pikachu can store in his electric pouch. Pikachu's electric power can be represented in binary form, and the number of Thunderbolt charges corresponds to the number of `1`s in the binary representation of each number.

Given an integer `n`, return an array `ans` of length `n + 1` such that for each number `i` `(0 <= i <= n)`, `ans[i]` is the number of Thunderbolt charges (`1`'s in the binary representation) Pikachu can store.

Write a function `thunderbolt_charges()` to calculate the number of Thunderbolt charges for each number from 0 to `n`.

```
def thunderbolt_charges(n):
    pass
```

Example Usage:

```
print(thunderbolt_charges(2))
print(thunderbolt_charges(5))
print(thunderbolt_charges(10))
```

Example Output:

```
[0, 1, 1]
Example 1 Explanation:
- 0 in binary is `0`, so Pikachu stores `0` Thunderbolt charges.
- 1 in binary is `1`, so Pikachu stores `1` Thunderbolt charge.
- 2 in binary is `10`, so Pikachu stores `1` Thunderbolt charge.

[0, 1, 1, 2, 1, 2]
Example 2 Explanation:
- 0 in binary is `0`, so Pikachu stores `0` Thunderbolt charges.
- 1 in binary is `1`, so Pikachu stores `1` Thunderbolt charge.
- 2 in binary is `10`, so Pikachu stores `1` Thunderbolt charge.
- 3 in binary is `11`, so Pikachu stores `2` Thunderbolt charges.
- 4 in binary is `100`, so Pikachu stores `1` Thunderbolt charge.
- 5 in binary is `101`, so Pikachu stores `2` Thunderbolt charges.

[0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2]
Example 3 Explanation:
- 0 in binary is `0`, so Pikachu stores `0` Thunderbolt charges.
- 1 in binary is `1`, so Pikachu stores `1` Thunderbolt charge.
- 10 in binary is `1010`, so Pikachu stores `2` Thunderbolt charges.
```

▶ 💡 **Hint: Dynamic Programming**

Problem 2: Charizard's Battle Power Calculation

Charizard is preparing for an important battle, and each of his moves contributes to his overall battle power. However, Charizard can either use a move to increase or decrease his power based on his strategy. Given a list of Charizard's moves, where each move has a power value, and a target battle power, your task is to determine how many different ways Charizard can combine his moves to exactly match the target battle power.

For each move in the list, you can choose to either add or subtract its power to a total sum.

For example, if `moves = [1, 2]`, one possible combination is "+1-2=-1".

Write a function `charizard_battle_power()` to calculate the number of different ways to combine Charizard's moves to reach the target battle power.

```
def charizard_battle_power(moves, target):
    pass
```

Example Usage:

```
print(charizard_battle_power([2, 2, 2], 2))
print(charizard_battle_power([1, 1], 0))
```

Example Output:

```
3
Example 1 Explanation:
There are 3 different ways for Charizard to combine his moves to achieve a total pow
+2 +2 -2 = 2
+2 -2 +2 = 2
-2 +2 +2 = 2

2
Example 2 Explanation:
There are 2 ways for Charizard to combine his moves to reach a total power of 0:
+1 -1 = 0
-1 +1 = 0
```

Problem 3: Team Rocket's Secret Pokémon Code

Team Rocket has intercepted a secret message encoded as a string of numbers, but they need your help to decode it! Each number corresponds to a letter:

- `"1"` -> 'A'
- `"2"` -> 'B'
- ...
- `"25"` -> 'Y'
- `"26"` -> 'Z'

There are multiple ways to decode some messages because some number codes overlap (e.g., `"2"` for 'B' and `"5"` for 'E' versus `"25"` for 'Y').

For example, the code `"11106"` could be decoded into:

- `"AAJF"` (with the grouping `[1, 1, 10, 6]`)
- `"KJF"` (with the grouping `[11, 10, 6]`)

Your task is to calculate how many different ways Team Rocket can decode the message. If it's impossible to decode, return `0`.

Write a function `decode_pokemon_code()` to find the number of ways the string can be decoded.

```
def decode_pokemon_code(s):
    pass
```

Example Usage:

```
print(decode_pokemon_code("12"))
print(decode_pokemon_code("226"))
print(decode_pokemon_code("06"))
```

Example Output:

```
2
Example 1 Explanation:
The code `"12"` could be decoded as `"AB"` or `"L"`.

3
Example 2 Explanation:
The code `"226"` could be decoded as `"BZ"`, `"VF"`, or `"BBF"`.

0
Example 3 Explanation:
The code `"06"` is invalid because `"06"` cannot be decoded (leading zeroes are not
```

Problem 4: Mewtwo's Genetic Fusion

Professor Oak has tasked you with a challenging experiment: fusing the DNA of two Pokémon, Mew and Ditto, to create a new species! The fusion process is governed by specific rules, where the genetic sequences of Mew (`dna1`) and Ditto (`dna2`) must be interwoven to form a new DNA sequence (`dna3`). Your goal is to determine if `dna3` can be formed by interleaving the sequences of `dna1` and `dna2` .

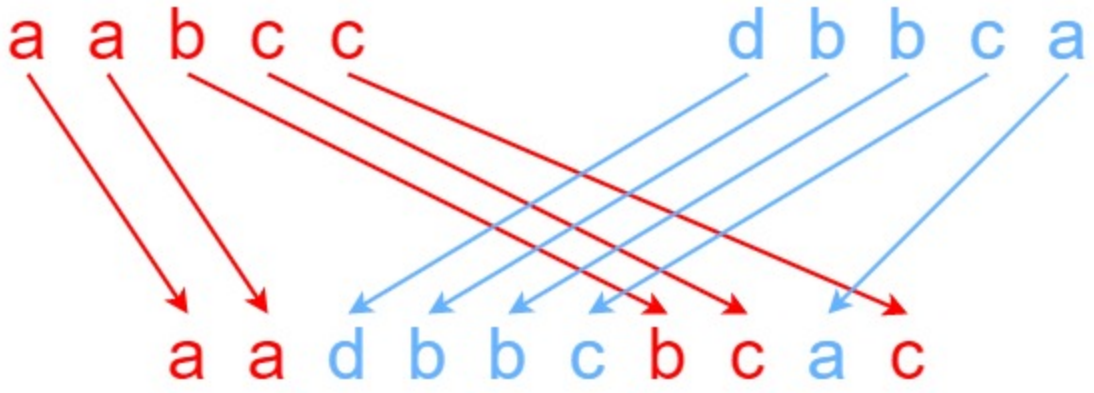
An interleaving of two genetic sequences `g1` and `g2` is a configuration where the sequences are divided into smaller substrings, and then combined alternately without changing the order of the substrings.

Help Professor Oak figure out if `dna3` can be obtained by interleaving `dna1` and `dna2` by implementing a function `genetic_fusion()` that returns `True` if the fusion is successful and `False` otherwise.

```
def genetic_fusion(dna1, dna2, dna3):
    pass
```

Example Usage:

Example 1:




```
print(genetic_fusion("aabcc", "dbbca", "aadbbcbcac"))
print(genetic_fusion("aabcc", "dbbca", "aadbbbaccc"))
print(genetic_fusion("", "", ""))
```

Example Output:

```
True
Example 1 Explanation:
One way to obtain `dna3` is:
Split `dna1` into "aa" + "bc" + "c", and `dna2` into "dbbc" + "a".
Interleaving the two sequences gives "aa" + "dbbc" + "bc" + "a" + "c" = "aadbbcbcac"
Since this forms `dna3`, the result is `True`.

False
Example 2 Explanation:
It is impossible to interleave `dna2` with any other string to form `dna3`.

True
Example 3 Explanation:
Empty strings can always be interleaved to form an empty string.
```

▶ 💡 **Hint: 2-D Dynamic Programming**

Problem 5: Gary's Pokédollar Trading Strategy II

Gary Oak has become experienced in the Pokémon trading market, and he's ready to make as many trades as possible to maximize his Pokédollars. Each day, the prices of Pokéballs fluctuate, and Gary wants to buy and sell Pokéballs at the best times. However, after making a sale, Gary needs a day to rest before he can make another trade. This rest period is called a cooldown.

You are given an array `prices` where `prices[i]` represents the value of a Pokémon trade on the `ith` day. Your task is to help Gary find the maximum profit he can achieve by completing as many trades as possible, while adhering to the following rules:

- Gary can make multiple transactions (buying and selling) but cannot hold multiple trades at the same time (he must sell a Pokéball before buying again).
- After selling a Pokéball, Gary must take a cooldown and skip trading the next day.

Write a function `max_pokedollar_profit()` that returns the maximum profit Gary can make.

```
def max_pokedollar_profit(prices):
    pass
```

Example Usage:

```
print(max_pokedollar_profit([1, 2, 3, 0, 2]))
print(max_pokedollar_profit([1]))
```

Example Output:

```
3
Example 1 Explanation:
Gary should buy on day 1, sell on day 2, rest on day 3, and then buy on day 4 and se
The transactions are `[buy, sell, cooldown, buy, sell]`, resulting in a total profit

0
Example 2 Explanation:
There are no profitable trades that can be made from the single-day prices.
```

Problem 6: Pikachu's Journey

Pikachu is navigating a maze of tall grass to reach the next Pokémon Center! The maze is laid out as an `m x n` grid, and Pikachu starts at the top-left corner (i.e., `grid[0][0]`) and must make his way to the bottom-right corner (i.e., `grid[m - 1][n - 1]`).

Pikachu can only move down or right at any point in time. Your task is to help Pikachu calculate the number of possible unique paths he can take to reach the Pokémon Center from the starting position.

Write a function `pikachu_unique_paths()` to calculate the total number of unique paths Pikachu can take.

```
def pikachu_unique_paths(m, n):
    pass
```

Example Usage:

```
print(pikachu_unique_paths(3, 7))
print(pikachu_unique_paths(3, 2))
```

Example Output:

```
28
Example 1 Explanation:
Pikachu can take 28 unique paths from the top-left corner to the bottom-right corner

3
Example 2 Explanation:
Pikachu can take 3 unique paths from the top-left corner to the bottom-right corner
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```

Close Section