Explanation of the Coin Change Problem

Explanation of the Given Code (Coin Change Problem)

The given code is an implementation of the "Coin Change" problem using Dynamic Programming (DP) to determine
the minimum number of coins needed to make a given amount.

Problem Statement:

Given a list of coin denominations (coins[]) and an integer amount, find the minimum number of coins needed

to make up the given amount. If it's not possible to make the amount using the given coins, return -1.

Code Breakdown:

1. Function Signature:

def coinChange(self, coins: List[int], amount: int) -> int:

- coins: List[int] -> A list of available coin denominations.
- amount: int -> The target amount to achieve.
- Returns the minimum number of coins needed to achieve the amount.

---

2. Initialization:

```
dp = [amount + 1] * (amount + 1)

dp[0] = 0
```

- dp[i] represents the minimum number of coins needed to achieve amount i.

- The list dp is initialized to amount + 1 to represent an impossibly high value.

- dp[0] = 0: Base case - No coins are needed to achieve amount 0.

---

3. Dynamic Programming Loop:

```
for a in range(1, amount + 1):

    for c in coins:

        if a - c >= 0:

            dp[a] = min(dp[a], 1 + dp[a - c])
```

- The outer loop iterates through each amount from 1 to amount.

- The inner loop goes through each available coin.

- The condition ensures the coin value does not exceed the current amount.

- The recurrence relation:

```
  dp[a] = min(dp[a], 1 + dp[a - c])
```

---

4. Final Decision:

return dp[amount] if dp[amount] != amount + 1 else -1

- If dp[amount] is still set to amount + 1, it means no valid combination of coins was found, so return -1.

- Otherwise, return the minimum number of coins required.

Example Walkthrough:

Example 1:

coins = [1, 2, 5]

amount = 11

Expected Output: 3 (as 5 + 5 + 1 = 11)

Time and Space Complexity:

- **Time Complexity:** O(N × M)

- **Space Complexity:** O(N)

Edge Cases to Consider:

1. No coins available -> Output: -1

2. Amount is 0 -> Output: 0

3. Impossible to make the amount -> Output: -1

4. Exact denomination match -> Output: 1

Final Thoughts:

This approach efficiently solves the coin change problem using dynamic programming by breaking down the problem

into subproblems and storing results to avoid redundant computations.