2240. Number of Ways to Buy Pens and Pencils

Medium Math Enumeration

Problem Description

In this problem, you are tasked with finding out the number of distinct ways you can use a given amount of money to purchase pens and pencils. Each pen costs `cost1`, and each pencil costs `cost2`. You can decide to buy any number of pens and pencils including none, as long as the total cost does not exceed the amount of money you have, which is `total`. The problem requires you to return the total number of different combinations of pens and pencils that you can buy.

Intuition

The key to solving this problem is to realize that for every number of pens you buy, there is a fixed number of pencils you can afford with the remaining money. This is a classic example of a counting problem where you iterate over one variable and directly compute the associated second variable's possible values.

To tackle the problem, you can start by iterating over the number of pens you can afford. For each possible quantity of pens, you calculate how much money would be left (`total - cost1 * number_of_pens`). Then, determine the maximum number of pencils you can buy with that remaining amount of money. The number of ways to buy pencils for each fixed amount of pens is just one more than the maximum number of pencils, as you can choose to buy from zero to that maximum number. The `+1` accounts for the option of buying no pencils at all.

For example, if you have 10 dollars, and pens cost 5 dollars each and pencils cost 1 dollar, you can buy 0, 1, or 2 pens (which cost 0, 5, or 10 dollars respectively). If you buy 0 pens, you can afford 0 to 10 pencils; if you buy 1 pen, you can afford 0 to 5 pencils, and if you buy 2 pens, you can afford no pencils. So in total, there are 11 (for 0 pens) + 6 (for 1 pen) + 1 (for 2 pens) = 18 distinct ways to buy pens and pencils.

The Python code in the solution section iterates through the number of pens (`x`) from 0 up to the maximum number you can afford with `total` money. For each amount of pens, it computes how many pencils (`y`) could be bought with the remaining money and sums these up to find the answer.

Solution Approach

The solution to the problem follows a straightforward brute-force approach due to the simplicity of the iteration required. Here's a step-by-step explanation of the `waysToBuyPensPencils` function:

1. Initialize a variable `ans` to store the total number of ways to buy pens and pencils.
2. Use a `for` loop to iterate through the number of pens (`x`) that can be bought with the given total. The range is from 0 to the quotient of `total` divided by `cost1` (the cost of a pen), because you cannot buy more pens than what you can afford with the available money. The upper limit is inclusive, so we add 1 to include the scenario where we spend all money on pens.
3. For each number of pens, calculate the remaining money after buying `x` pens with `(total - (x * cost1))`.
4. Determine the number of pencils (`y`) that can be bought with that remaining money by dividing it by `cost2` (the cost of a pencil) and again add 1 to include the scenario where we choose not to buy any pencils.
5. Add the number of ways to buy pencils (`y`) to `ans` for each iteration.
6. After the loop finishes, `ans` contains the total number of distinct ways to buy pens and pencils, which is then returned.

The algorithm does not use any complex data structures as it's based on simple arithmetic operations and iteration. There are no patterns like dynamic programming or divide-and-conquer being used, as the problem doesn't have overlapping subproblems or a need to break down into smaller problems. The brute-force approach is efficient in this context because one variable (the number of pens) can be directly iterated over, and the second variable (the number of pencils) can be calculated instantly without additional iteration.

Here is the key portion of the Python function in code:

```
def waysToBuyPensPencils(self, total: int, cost1: int, cost2: int) -> int:
    ans = 0
    for x in range(total // cost1 + 1):
        y = (total - (x * cost1)) // cost2 + 1
        ans += y
    return ans
```

This method makes sure all the possible combinations of pens and pencils are counted without redundancy, and each combination is distinct as it results from a unique number of pens (`x`) bought each iteration.

Example Walkthrough

Let's consider a small example to illustrate the solution approach. Suppose you have `total = 7` dollars, `cost1 = 2` dollars per pen, and `cost2 = 3` dollars per pencil.

We want to determine the total number of distinct ways we can buy pens and pencils without exceeding the total of 7 dollars. Follow these steps:

1. Start with initializing the total number of combinations, `ans = 0`.
2. Now, iterate through the number of pens `x` you could buy, starting from 0. The maximum number of pens you can buy with 7 dollars is `7 // 2 = 3` (since you cannot buy half a pen), and you add 1 to this range for iteration to account for buying no pens at all. The loop iterates for `x = 0, 1, 2, 3`.
3. The following table shows the calculation for each iteration of `x`:

Number of Pens (x)	Spent on Pens	Remaining Money	Max Pencils (y)	Total Ways (y + 1)
0	0 * 2 = 0	7 - 0 = 7	7 // 3 = 2	2 + 1 = 3
1	1 * 2 = 2	7 - 2 = 5	5 // 3 = 1	1 + 1 = 2
2	2 * 2 = 4	7 - 4 = 3	3 // 3 = 1	1 + 1 = 2
3	3 * 2 = 6	7 - 6 = 1	1 // 3 = 0	0 + 1 = 1

   Note that for each number of pens, we add 1 to the max pencils `(y)` to account for the option of not buying any pencils at all.
4. Adding up the total ways from the last column of the table for each iteration gives `ans = 3 + 2 + 2 + 1 = 8`. So, there are 8 distinct ways to buy pens and pencils with 7 dollars.
5. The loop concludes, and the final answer, `ans = 8`, is returned. Therefore, with 7 dollars, you have 8 different combinations to buy pens that cost 2 dollars and pencils that cost 3 dollars.

This example aligns with the solution approach where every possibility to buy pens is considered, followed by a direct computation of the number of pencils that can be bought with the leftover money. The key portion of the Python code:

```
def waysToBuyPensPencils(total: int, cost1: int, cost2: int) -> int:
    ans = 0
    for x in range(total // cost1 + 1):
        y = (total - (x * cost1)) // cost2 + 1
        ans += y
    return ans
```

This function would indeed return 8 for this example, confirming our manual computation.

Solution Implementation

Python

```
class Solution:
    def waysToBuyPensPencils(self, total: int, pen cost: int, pencil cost: int) -> int:
        # Initialize the answer variable to keep track of the number of ways to buy pens and pencils.
        num_ways = 0

        # Calculate the maximum possible number of pens that can be bought with the total amount.
        max_pens = total // pen_cost

        # Iterate over the range of possible pens that can be bought (from 0 to max_pens inclusive).
        for num_pens in range(max_pens + 1):
            # Calculate the remaining total after buying num_pens pens.
            remaining_total = total - (num_pens * pen_cost)

            # Compute the number of pencils that can be bought with the remaining total.
            num_pencils = (remaining_total // pencil_cost) + 1

            # Add the number of ways to buy pencils with the given number of pens to the answer.
            num_ways += num_pencils

        # Return the total number of ways to buy pens and pencils.
        return num_ways
```

Java

```
class Solution {
    // Method to calculate the number of ways to purchase pens and pencils with a given total amount.
    public long waysToBuyPensPencils(int total, int costPerPen, int costPerPencil) {
        long numberOfWays = 0; // Initialize the count of ways to 0.

        // Loop through all possible quantities of pens we can buy within the budget.
        for (int numOfPens = 0; numOfPens <= total / costPerPen; ++numOfPens) {
            // Calculate the remaining budget after buying the pens.
            int remainingBudget = total - numOfPens * costPerPen;
            // Calculate the number of pencils we can buy with the remaining budget.
            int numOfPencils = remainingBudget / costPerPencil + 1;
            // Add the number of ways we can purchase pencils with the remaining budget
            // to the total count of purchase ways.
            numberOfWays += numOfPencils;
        }

        return numberOfWays; // Return the total count of ways.
    }
}
```

C++

```
class Solution {
public:
    // Function to calculate the number of ways to buy pens and pencils
    // given a total amount of money, cost of a pen, and cost of a pencil.
    long long waysToBuyPensPencils(int totalAmount, int costOfPen, int costOfPencil) {
        long long numberOfWays = 0; // Initialize the number of ways to 0

        // Iterate through all possible quantities of pens we can buy.
        for (int pens = 0; pens <= totalAmount / costOfPen; ++pens) {
            // Calculate the remaining amount after buying 'pens' number of pens.
            int remainingAmount = totalAmount - pens * costOfPen;
            // Calculate the number of pencils we can buy with the remaining amount.
            int pencils = remainingAmount / costOfPencil + 1; // +1 because we can also choose to buy 0 pencils.
            // Add the possible number of pencils to the total number of ways.
            numberOfWays += pencils;
        }

        return numberOfWays; // Return the total number of ways to buy pens and pencils.
    }
};
```

TypeScript

```
function waysToBuyPensPencils(total: number, costOfPen: number, costOfPencil: number): number {
    // Initialize the count of ways to 0.
    let numberOfWays = 0;

    // Iterate over the possible numbers of pens that can be bought within the total budget.
    for (let numPens = 0; numPens <= Math.floor(total / costOfPen); ++numPens) {
        // Calculate the remaining budget after buying the pens.
        const remainingBudget = total - numPens * costOfPen;

        // Calculate the maximum number of pencils that can be bought with the remaining budget.
        const numPencils = Math.floor(remainingBudget / costOfPencil) + 1;

        // Add the number of ways to buy pencils with the remaining budget to the total count.
        numberOfWays += numPencils;
    }

    // Return the total count of ways to buy pens and pencils.
    return numberOfWays;
}
```

```
class Solution:
    def waysToBuyPensPencils(self, total: int, pen cost: int, pencil cost: int) -> int:
        # Initialize the answer variable to keep track of the number of ways to buy pens and pencils.
        num_ways = 0

        # Calculate the maximum possible number of pens that can be bought with the total amount.
        max_pens = total // pen_cost

        # Iterate over the range of possible pens that can be bought (from 0 to max_pens inclusive).
        for num_pens in range(max_pens + 1):
            # Calculate the remaining total after buying num_pens pens.
            remaining_total = total - (num_pens * pen_cost)

            # Compute the number of pencils that can be bought with the remaining total.
            num_pencils = (remaining_total // pencil_cost) + 1

            # Add the number of ways to buy pencils with the given number of pens to the answer.
            num_ways += num_pencils

        # Return the total number of ways to buy pens and pencils.
        return num_ways
```

Time and Space Complexity

Time Complexity

The time complexity of the given code is primarily determined by the `for` loop, which iterates based on the result of the division `total // cost1 + 1`. This is because for each iteration, it calculates the maximum number of pencils that can be bought after buying `x` pens.

For each `x` (number of pens), there is a constant-time operation to calculate the value of `y` (the `total - (x * cost1)) // cost2 + 1`), which represents the number of ways you can buy pencils given the remaining amount of money after buying `x` pens. Therefore, the loop runs in `O(total // cost1 + 1)` time.

The `+1` in `total // cost1 + 1` is constant and doesn't affect the overall order of growth, so it can be dropped for the Big O notation. Hence, the time complexity is `O(total // cost1)` which can also be written as `O(total / cost1)`.

Space Complexity

The space complexity of the function is constant, `O(1)`, because it only uses a fixed number of variables (`ans`, `x`, and `y`) that do not depend on the size of the input.