2214. Minimum Health to Beat Game

Medium Greedy Array Prefix Sum

Leetcode Link

Problem Description

In this game, you're facing a series of `n` levels where each level has a certain amount of damage associated with it, provided in the `damage` array. The `damage[i]` denotes how much health you will lose when you complete level `i`. Additionally, you have armor that can be used once to absorb some damage, up to the value of `armor`. However, the armor can't prevent more damage than the amount it's worth. Your goal is to find the minimum starting health that enables you to finish all levels. The important points to consider are:

- You must go through the levels in sequence from `0` to `n - 1`.
- Your health should always be greater than `0` to continue the game.
- The armor is a one-time use mechanism that can absorb damage up to its value on any one level of your choice.

The challenge is to strategically use the armor on a level to minimize the health you must start with in order to complete all the levels.

Intuition

To solve this problem, we take the following steps:

1. Calculate the total amount of damage that will be taken when passing all levels. This can be found by summing up all values in the `damage` array.
2. Decide when to use the armor. Ideally, to minimize the starting health, the armor should be used on the level where it will prevent the most damage. This implies that the armor should be used on the level with the maximum `damage` value.
3. Calculate the effective armor value. Since you cannot absorb more damage than the `armor` value, the effective protection from the armor is the minimum of the maximum damage among all levels and the `armor` value itself.
4. Subtract this calculated effective armor value from the total damage to obtain the actual damage you need to absorb with your health.
5. Since your health must always be greater than `0`, simply add `1` to the total actual damage to determine the minimum health needed to start the game.

Bringing all this together, the solution efficiently calculates the minimum health needed to beat the game with the given constraints.

Solution Approach

The implementation of the provided solution uses simple calculation and manipulation of the data structures given, which are a list for the `damage` and an integer for `armor`.

1. **Summing Damage**: The `sum()` function is used to calculate the total sum of the elements within the `damage` array, which represents the total damage you'll incur if you don't use the armor.

```
total_damage = sum(damage)
```

2. **Optimal Use of Armor**: To make the best use of the armor, we figure out the maximum damage that could be absorbed. This is found by identifying the maximum damage from the levels using `max(damage)`. However, since the armor cannot absorb more than its own value, we use `min(max(damage), armor)` to determine the actual damage that will be mitigated by the armor.

```
effective_armor_protection = min(max(damage), armor)
```

3. **Actual Damage Incurred**: Now, we subtract the effective armor protection (the actual damage the armor can absorb) from the total calculated damage.

```
actual_damage_incurred = total_damage - effective_armor_protection
```

4. **Minimum Health Calculation**: Ultimately, since health must always be above `0`, the minimum health needed to beat the game, factoring in the optimal use of armor, is found by adding `1` to the `actual_damage_incurred`.

```
min_health = actual_damage_incurred + 1
```

The above steps are compactly expressed in the given solution:

```
def minimumHealth(self, damage: List[int], armor: int) -> int:
    return sum(damage) - min(max(damage), armor) + 1
```

This algorithm is O(n) given that finding the maximum in an array `damage` and summing its elements both take O(n) time. No additional data structures are used besides the input, making it a space-efficient approach as well.

Note: In the code implementation, these steps are combined into one line for efficiency and brevity, but they conceptually represent the algorithm's workflow described above.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach:

Suppose we have the following `damage` array and `armor` value:

- `damage = [5, 10, 7]`
- `armor = 10`

According to our game's rules and solution approach, here's what we do:

1. **Summing Damage**: We add up all the damage values in the array:

```
total_damage = 5 + 10 + 7 = 22
```

2. **Optimal Use of Armor**: The maximum damage in a single level is `10`. Since our armor value is equal to the maximum damage, armor can completely absorb the damage from this level. So, the effective armor protection is:

```
effective_armor_protection = min(10, 10) = 10
```

3. **Actual Damage Incurred**: Now, we calculate the actual damage you incur by subtracting the armor's protection from the total damage:

```
actual_damage_incurred = 22 - 10 = 12
```

4. **Minimum Health Calculation**: To ensure your health is always greater than `0`, you need to start with one more health point than the actual damage incurred. Hence, the minimum health required would be:

```
min_health = 12 + 1 = 13
```

In this example, you would need a minimum of `13` health to complete all levels of the game, using your armor optimally to fully absorb the damage from the level with the highest damage.

Python Solution

```python
class Solution:
    def minimumHealth(self, damage: List[int], armor: int) -> int:
        """
        Calculate the minimum health required to survive a series of damage hits,
        given that armor can completely absorb the damage from one hit.

        :param damage: List[int] indicating the damage value of each hit
        :param armor: int representing the value of the armor
        :return: int minimum health needed to survive all hits
        """

        # Calculate total damage that would be taken without armor
        total_damage = sum(damage)

        # Determine the maximum single hit damage
        max_damage = max(damage)

        # Calculate the effective damage reduction by armor, capped at the max hit
        # This ensures that the armor does not absorb more than the heaviest hit
        effective_armor = min(max_damage, armor)

        # Calculate the minimum health required:
        # It's the total damage minus the effective armor value, plus one.
        # Adding one because health needs to be at least 1 to survive.
        minimum_health = total_damage - effective_armor + 1

        return minimum_health

```

Java Solution

```java
class Solution {
    public long minimumHealth(int[] damage, int armor) {
        // Initialize the sum of all damage values to 0
        long totalDamage = 0;
        // Initialize the variable to hold the maximum single hit damage
        int maxDamage = damage[0];

        // Iterate over the damage array to calculate the total damage
        // and find the maximum single hit damage
        for (int currentDamage : damage) {
            totalDamage += currentDamage;
            maxDamage = Math.max(maxDamage, currentDamage);
        }

        // Calculate the minimum health required to survive all the damage,
        // taking into account the armor which can absorb damage not exceeding the value of the strongest hit.
        // Add 1 to ensure that the health is strictly above 0 after taking all damage.
        return totalDamage - Math.min(maxDamage, armor) + 1;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Calculate the minimum health required to defeat the monsters.
    // damage - list of integers representing the damage of each monster.
    // armor - an integer representing the maximum damage the armor can absorb.
    long long minimumHealth(vector<int>& damage, int armor) {
        long long totalDamage = 0; // To store the sum of all damages.
        int maxDamage = damage[0]; // Initialize maxDamage with the first element.

        // Iterate over the damage array to find the sum of all damages
        // and the maximum damage dealt by any single monster.
        for (int dmg : damage) {
            totalDamage += dmg;     // Accumulate the total damage.
            maxDamage = max(maxDamage, dmg); // Keep track of the max damage.
        }

        // Calculate and return the minimum health needed:
        // Subtract the smaller of the max damage or the armor from total damage
        // and add 1 (since the health must be at least 1 to survive).
        return totalDamage - min(maxDamage, armor) + 1;
    }
};

```

Typescript Solution

```typescript
// Function to calculate the minimum health required to withstand damage with the potential armor reduction applied
function minimumHealth(damage: number[], armor: number): number {
    let totalDamage = 0;  // Initialize the total damage taken to zero
    let maxDamage = 0;    // Initialize the maximum single hit damage to zero

    // Iterate over the array of damage values
    for (const singleDamage of damage) {
        // Update the maximum single hit damage if the current one is greater
        maxDamage = Math.max(maxDamage, singleDamage);
        // Sum up the total damage taken
        totalDamage += singleDamage;
    }

    // Calculate minimum health required:
    // Total damage minus the lesser of max damage or armor (the armor can only reduce up to the max damage taken in a single hit)
    // We add 1 because health must be at least 1 more than the remaining damage after armor reduction
    return totalDamage - Math.min(maxDamage, armor) + 1;
}

```

Time and Space Complexity

- Time Complexity: The time complexity of the method `minimumHealth` is `O(n)`, where `n` is the length of the `damage` list. This is because the functions `sum(damage)`, `min(damage)`, and `max(damage)` each require traversing the entire list once. No other operations inside the method have a higher time complexity, so the overall complexity does not exceed linear time.

- Space Complexity: The space complexity of the method `minimumHealth` is `O(1)`. This is because the space required does not depend on the size of the input list `damage`. The memory used is constant as we only need a fixed amount of space to store intermediate results such as the sum of the `damage` list, the minimum value, the maximum value, and the final health calculation.