1953. Maximum Number of Weeks for Which You Can Work

Medium Greedy Array Leetcode Link

Problem Description

In this problem, we are given `n` projects, each having a certain number of milestones that need to be completed. Each `milestones[i]` represents the number of milestones for the `i`th project. The goal is to work on these projects under two constraints:

1. Each week, we complete exactly one milestone from one project. We must work every week without skipping any.
2. We are not allowed to work on milestones from the same project for two consecutive weeks.

We are required to keep working on the projects until either all the milestones are completed or it becomes impossible to continue without breaking the rules. The task is to calculate the maximum number of weeks we can work on these projects without violating the given rules.

Intuition

To solve the problem, let's consider the following points:

- If the number of milestones for one project is more than the sum of milestones of all other projects plus one (i.e., `max(milestones) > sum(milestones) - max(milestones) + 1`), we will inevitably end up violating the rule about not working on the same project for two consecutive weeks. This is because after finishing all the milestones from other projects, we would still have at least two milestones left from the project with the maximum milestones, which would force us to work on the same project back-to-back.

- If the maximum number of milestones is less than or equal to the sum of the rest of the milestones plus one, we can always alternate between the project with the most milestones and the other projects.

With these insights, we arrive at the core of the solution:

1. If `max(milestones)` (the largest number of milestones from any single project) is greater than the total milestones of all other projects plus one, we can work for `2 * (total milestones of all other projects) + 1` weeks, because after that we will be left only with milestones from the project with the maximum milestones, which we cannot work on consecutively.
2. If the maximum milestones are less than or equal to the sum of all other milestones plus one, we can complete all the work without having any leftover milestones, working for `sum(milestones)` weeks.

By using this strategy, our code neatly captures the maximum number of weeks we can work.

Solution Approach

The solution is relatively straightforward and does not require complex data structures or algorithms. Here's the step-by-step approach:

1. First, we identify the project with the maximum number of milestones by using Python's `max()` function on our list of milestones. This gives us the maximum value `mx`.

2. We also compute the sum of all milestones across all projects using the `sum()` function. This total is stored in the variable `s`.

3. We then calculate the "rest" by subtracting the maximum number of milestones from the total sum, `rest = s - mx`. This represents the total milestones from projects other than the one with the maximum milestones.

4. We use an `if` condition to check whether the maximum number of milestones is greater than the sum of milestones from other projects plus one (`mx > rest + 1`). If true, this means we will eventually face a situation where we have to work on the same project for two consecutive weeks, which breaks the rules. In this case, the maximum number of weeks we can work is `rest * 2 + 1`, because we can alternate between projects until the non-maximum projects are completed, leaving just one week left to work on the max project before breaking the rule.

5. If the condition is false, it implies we can finish all milestones without breaking the rules. Hence, we can work for `s` weeks, which represents the total sum of milestones.

The pattern used here is primarily based on mathematical reasoning rather than applying specific algorithms or data structures. The key is to quickly calculate the total sum and the largest element and make a decision based on the constraints given in the problem.

The Python code provided encapsulates this logic succinctly in two lines, making use of Python's list operations to find the required max and sum values and performing a simple conditional operation to arrive at the result.

Example Walkthrough

Let's consider an example with `n` projects with their respective milestones: `milestones = [3, 2, 2]`. In this case, the project with the maximum number of milestones (project 1) has 3 milestones.

Following the step-by-step solution approach:

1. Identify the project with the maximum number of milestones:
   - `mx = max(milestones)` gives us `mx = 3`.
2. Compute the sum of all milestones across all projects:
   - `s = sum(milestones)` gives us `s = 3 + 2 + 2 = 7`.
3. Calculate the milestones from the other projects excluding the max project:
   - `rest = s - mx` gives us `rest = 7 - 3 = 4`.
4. Check if we can avoid working on the same project for two consecutive weeks:
   - We evaluate `mx > rest + 1`, which is `3 > 4 + 1`. This condition is false.
5. Since the condition is false, it implies we will not have to work on the same project for two consecutive weeks at any point. Therefore, we can work on all milestones without breaking the rules:
   - The maximum number of weeks we can work is `s` weeks, so the answer is `7` weeks.

In this scenario, a possible sequence could be working on projects in the following order over 7 weeks: `1, 2, 1, 3, 1, 2, 3`. Each time we work on the project with the maximum milestones (project 1), we alternate it with work on the other projects, ensuring we do not violate the constraint of working on the same project in consecutive weeks. By following the described approach, we've determined that we can work for a total of 7 weeks.

Python Solution

```python
from typing import List  # Import List from the typing module for type hinting

class Solution:
    # Function to calculate the maximum number of weeks of work
    def number_of_weeks(self, milestones: List[int]) -> int:
        max_milestone = max(milestones)  # Get the maximum value from milestones list
        total_sum = sum(milestones)      # Calculate the sum of all milestones
        rest = total_sum - max_milestone # Calculate the sum of all other milestones except the maximum one

        # If the maximum milestone is more than the sum of other milestones + 1,
        # the maximum number of weeks we can work would be (rest * 2) + 1
        # (work on other projects and on the max one alternatively).
        if max_milestone > rest + 1:
            return (rest * 2) + 1
        else:
            # Otherwise, we can work on all projects without any project taking
            # an entire additional week on its own.
            return total_sum

# Example usage:
# sol = Solution()
# result = sol.number_of_weeks([1, 2, 3])
# print(result)  # This will print 6, as all milestones can be completed without restrictions

```

Java Solution

```java
class Solution {
    public long numberOfWeeks(int[] milestones) {
        int maxMilestone = 0; // Initialize variable to store the maximum value of milestones
        long totalMilestonesSum = 0; // Initialize a variable to store the sum of all milestones

        // Iterate through each milestone and calculate the total sum and max milestone
        for (int milestone : milestones) {
            totalMilestonesSum += milestone; // Add the current milestone to the total sum
            maxMilestone = Math.max(maxMilestone, milestone); // Update max milestone if the current one is greater
        }

        // Calculate the sum of all milestones except the maximum one
        long rest = totalMilestonesSum - maxMilestone;

        // If the maximum milestone is more than the sum of the rest plus one,
        // then return twice the sum of the rest plus one as maximum weeks
        // This ensures that we cannot complete the maximum milestone if it's too large compared to the others
        if (maxMilestone > rest + 1) {
            return rest * 2 + 1;
        } else {
            // Otherwise, return the total sum of all milestones meaning all milestones can be completed
            return totalMilestonesSum;
        }
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    long long numberOfWeeks(vector<int>& milestones) {
        // Find the largest milestone.
        int maxMilestone = *max_element(milestones.begin(), milestones.end());

        // Calculate the total sum of all milestones.
        long long sum = accumulate(milestones.begin(), milestones.end(), 0LL);

        // Calculate the total sum of all milestones excluding the largest one.
        long long sumWithoutMax = sum - maxMilestone;

        // If the largest milestone is greater than the sum of the rest plus one,
        // we can only work on each project without exceeding the maximum until we
        // complete projects that amount to double the rest (evenly distributed)
        // and then finish one more week of work on the largest one.
        if (maxMilestone > sumWithoutMax + 1) {
            return sumWithoutMax * 2 + 1;
        } else {
            // Otherwise, we can complete all weeks of work without any problem.
            // In this case, sum of all milestones equals total number of weeks.
            return sum;
        }
    }
};

```

Typescript Solution

```typescript
function numberOfWeeks(milestones: number[]): number {
  // Find the largest milestone.
  const maxMilestone = Math.max(...milestones);

  // Calculate the total sum of all milestones.
  const sum = milestones.reduce((a, b) => a + b, 0);

  // Calculate the total sum of all milestones, excluding the largest one.
  const sumWithoutMax = sum - maxMilestone;

  // Check if the largest milestone is greater than the sum of the rest plus one.
  // If so, we are limited by how many weeks we can work without exceeding the maximum.
  if (maxMilestone > sumWithoutMax + 1) {
    // Return the maximum possible number of weeks we can work without exceeding the largest milestone.
    // This is twice the sum of the other milestones plus one week on the largest project.
    return sumWithoutMax * 2 + 1;
  } else {
    // Otherwise, we can complete all weeks of work on all projects without any issues.
    // In this case, the sum of all milestones equals the total number of possible weeks of work.
    return sum;
  }
}

```

Time and Space Complexity

Time Complexity

The time complexity of the solution is `O(N)`, where `N` is the number of elements in the `milestones` list. This is because the algorithm needs to iterate through all elements twice: once to calculate the sum `s`, and once to find the maximum value `mx` in the list.

Space Complexity

The space complexity of the solution is `O(1)`. It only uses a constant amount of extra space to store variables `mx`, `s`, and `rest`, regardless of the size of the input list `milestones`.