492. Construct the Rectangle

Easy Math

Leetcode Link

Problem Description

A web developer is tasked with designing a rectangular web page, where the challenge lies in determining the length (`L`) and width (`W`) of the page. These dimensions must meet three specific conditions:

1. The rectangular web page's area, that is, `L * W`, must be exactly equal to a given target area.
2. The width (`W`) should not exceed the length (`L`), which means the layout should either be a square or a landscape-oriented rectangle (`L >= W`).
3. The absolute difference between the length and width should be minimized to make the dimensions as close to a square as possible, which usually is aesthetically pleasing and often a desired trait in web page design.

The final output should be presented as an array `[L, W]`, indicating the length and width of the website's page, respectively.

Intuition

To solve this problem, we can leverage the mathematical fact that for any given rectangular area, if we're trying to minimize the difference between length and width, the best approach is to aim for a shape that is as close to a square as possible. This is because, for a fixed area, a square has the smallest perimeter out of all possible rectangles.

By that logic, we should start looking for the optimal width `W` by considering the largest possible square that could fit in the given area, which would be a square whose sides are equal to the square root of the area. However, since the area may not be a perfect square, we look for the greatest divisor of the area that is less than or equal to the square root because this would give us the width that is closest to the length, thereby minimizing their difference.

Here's how the code reflects this intuition:

1. Compute an initial width `W` by taking the integer square root of the area, since decimals aren't permissible for the height or width of a web page.

2. Verify if this width is indeed a divisor of the area by checking the remainder of the division of the area by `W`.

3. If it's not, decrement the width `W` by 1, and repeat this step until a divisor is found.

4. Once the proper width `W` is found (where `area % W == 0`), calculate the corresponding length `L` by dividing the area by `W`.

5. Return the pair `[L, W]` as the result, with `L` being the larger value and `W` being the smaller or equal value.

This approach ensures that the shape of the rectangle will be as close to a square as possible, adhering to the web developer's needs.

Solution Approach

The implementation of the solution is quite straightforward, with no need for advanced data structures or complex algorithms. Here is the breakdown of the code provided in the solution:

```python
class Solution:
    def constructRectangle(self, area: int) -> List[int]:
        w = int(sqrt(area))  # Starting from the largest possible square
        while area % w != 0: # Continue until a divisor is found
            w -= 1           # Decrement width by 1
        return [area // w, w] # Calculate length and return result
```

1. **Finding the Width (`W`)**: The initial width is guessed by taking the integer part of the square root of the area (`int(sqrt(area))`). This is based on the idea that for a rectangle with a given area, the configuration closest to a square will minimize the difference between length and width.

2. **Validating Width (`W`)**: Since the initial guess may not be the exact width (unless the area itself is a perfect square), a `while` loop verifies if the current width exactly divides the area (`area % w == 0`).

3. **Iterative Approach**: If the current width isn't a divisor of the area (meaning `area % w != 0`), decrement the width (`w`) by one, and recheck. Continue this process until you find the largest divisor of the area that is less than or equal to the square root of the area.

4. **Calculating Length (`L`)**: Once a suitable width (`W`) is found, calculate the corresponding length by dividing the area by this width (`area // w`).

5. **Returning the Result**: The resulting `L` and `W` are packed into a list in the correct order (`[L, W]`), since by definition, `L` must be equal to or larger than `W`.

The simplicity of this solution lies in the fact that for a given `area`, as `W` decreases, `L` increases proportionally because `area = L * W`. Thus, starting from the square root of the area and going downwards ensures we're approaching the smallest potential difference between the `L` and `W`.

The use of iteration and simple arithmetic operations make the algorithm effective and efficient with a time complexity of approximately $O(\sqrt{n})$, where n is the given area. It stops as soon as a divisor is found, and no additional space is necessary, hence space complexity is O(1).

Example Walkthrough

Let's consider a small example where the target area of our web page is `20`. We want to find the dimensions `[L, W]` that satisfy our three conditions:

1. The area equals `20` (i.e., `L * W = 20`).
2. The width `W` is less than or equal to the length `L` (`L >= W`).
3. The absolute difference between length and width is minimized.

Following the solution approach:

1. We start by finding the initial guess for the width `W` through the square root of the area. The square root of `20` is approximately `4.47`. Taking the integer part gives us `4`.

2. We must verify if `4` is a divisor of `20` by checking if `20 % 4 == 0`. Indeed it is, as `20` divided by `4` leaves no remainder.

3. Since `4` is a divisor, there's no need for the iterative approach to decrement the width and check for other possible divisors.

4. We calculate the corresponding length `L` by dividing the area by the width: `20 // 4 = 5`.

5. We then pack our dimensions `[L, W]` into an array. In this case, `[5, 4]`.

The dimensions of the web page that satisfy the given requirements are a length of `5` and a width of `4`, which gives us the smallest possible difference between length and width (minimizing the absolute difference), and also `L * W` equals the target area of `20`. Hence, the final output for this example is `[5, 4]`.

This example illustrates on a smaller scale, how the algorithm effectively finds the ideal dimensions for the web page by starting from the largest possible square root and verifying if it's an exact divisor of the given area. If not, it systematically explores smaller values of width to find the optimal dimensions, satisfying all three conditions.

Python Solution

```python
from math import sqrt
from typing import List

class Solution:
    def constructRectangle(self, area: int) -> List[int]:
        # Find the square root of the given area and cast it to an integer.
        # This is the starting point to search for the width because the largest
        # possible width (w) that's less than or equal to the length (l)
        # is when l == w (i.e., a square).
        width = int(sqrt(area))

        # Iterate by decreasing width to find the largest possible width
        # that evenly divides the given area (i.e., area % width == 0).
        # This is done to ensure the length is at least as large as the width,
        # as per problem constraints.
        while area % width != 0:
            width -= 1

        # Calculate the corresponding length by dividing the area by width.
        length = area // width

        # Return the dimensions [length, width] as a list, where length >= width.
        return [length, width]

```

Java Solution

```java
class Solution {

    // Method to construct a rectangle with a given area that is as close to a square as possible
    public int[] constructRectangle(int area) {
        // Start with the largest possible square root of the area as the width
        int width = (int) Math.sqrt(area);

        // Decrease the width until we find a value that perfectly divides the area
        while (area % width != 0) {
            --width;
        }

        // Since width is the smaller dimension, it is placed second in the array
        // The corresponding length is calculated by dividing the area by width
        return new int[] {area / width, width};
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <cmath> // Include cmath for using the sqrt function

class Solution {
public:
    // This function constructs a rectangle of the given area.
    // The rectangle's length is greater than or equal to its width,
    // and the difference between length and width is minimized.
    vector<int> constructRectangle(int area) {
        // Use integer sqrt function to find the square root of the area,
        // which is the best starting point for the width (w).
        int width = static_cast<int>(sqrt(area));

        // Loop to find the largest width which divides the area with no remainder.
        while (area % width != 0) {
            --width; // Reduce the width by one and check again.
        }

        // Width found, calculate length by dividing the area by width.
        // Return the pair {length, width} as the result.
        return {area / width, width};
    }
};

```

Typescript Solution

```typescript
function constructRectangle(area: number): number[] {
    // Find the square root of the area, which serves as the starting point for the possible width of the rectangle.
    let width: number = Math.floor(Math.sqrt(area));

    // Loop to find the greatest width that evenly divides the area
    while (area % width !== 0) {
        width--; // Decrement the width and verify again
    }

    // Once the width is found, calculate the length by dividing the area by this width
    const length: number = area / width;

    // Return an array containing the length and width of the rectangle
    return [length, width];
}

```

Time and Space Complexity

Time Complexity

The time complexity of the given code largely depends on the number of iterations in the while loop. Initially, it finds the square root of the `area`, which is executed in constant time, i.e., `O(1)`. After that, it decrements `w` until it finds a divisor of the `area`. In the worst case, this will occur when the `area` is a prime number, and `w` will be decremented down to 1, which would make the total number of iterations approximately `sqrt(area)` in the worst case. Therefore, the time complexity is `O(sqrt(area))`.

Space Complexity

The space complexity is `O(1)` because only a fixed amount of extra space is used, regardless of the input size. The variables `w` and the array that is returned do not increase with the size of the input `area`.