3000. Maximum Area of Longest Diagonal Rectangle

Easy Array

Problem Description

The given problem requires us to find the area of a rectangle with the longest diagonal among all rectangles in the provided 2D integer array `dimensions`. Each element of this array is itself an array of two elements; the first element represents the length and the second represents the width of a rectangle. If multiple rectangles have diagonals of the same maximum length, we need to find the area of the rectangle that has the maximum area among them.

Intuition

The primary step is to understand that the diagonal of a rectangle can be calculated using the Pythagorean theorem, where the diagonal is the hypotenuse of the right-angled triangle formed by the length and width of the rectangle. To find the longest diagonal, we will square the lengths and widths of the rectangles to avoid working with floating-point numbers that arise from square roots, which simplifies comparisons.

The solution iterates over each rectangle, calculating the square of its diagonal (`l**2 + w**2`). We also track the maximum diagonal length found so far (`mx`) and the area of the corresponding rectangle (`ans`). If a rectangle's diagonal squared is greater than the current maximum, we update both `mx` and `ans`. If it's equal to the current maximum, we only update `ans` if the rectangle's area is larger than the current largest area.

Solution Approach

The implementation of the solution adopts a straightforward approach consistent with the intuition previously described:

1. We initialize two variables, `ans` and `mx`, both set to `0`. `ans` will hold the maximum area found, and `mx` will hold the square of the longest diagonal found so far.
2. We then loop through each rectangle's dimensions in the `dimensions` list. For each rectangle, we calculate `t`, the square of the length of the diagonal, using the formula `l**2 + w**2` where `l` is the length and `w` is the width of the rectangle. This step is equivalent to applying the Pythagorean theorem to find the diagonal length without taking the square root.
3. Within the loop, we perform a check to see if the calculated `t` is greater than our current maximum `mx`. If it is, we have found a new rectangle with a longer diagonal, so we update `mx` to this new maximum, and we update `ans` to the area of the current rectangle, which is `l * w`.
4. If `t` is equal to the current `mx`, it means we have found another rectangle with a diagonal equal to the longest one seen so far. Since the problem requires us to return the area of the rectangle with the maximum area in such cases, we update `ans` with the larger of its current value or the current rectangle's area.
5. After the loop has finished executing, `ans` contains the correct maximum area, which we return.

This approach ensures that the algorithm only passes through the array of rectangles once, making it efficient. The algorithm's time complexity is O(n), where n is the number of rectangles given.

In summary, this solution uses simple iteration and comparison, with no need for complex data structures or additional memory beyond the two tracking variables, making it a straightforward but effective approach to solve the problem.

Example Walkthrough

Let's consider a small example with the `dimensions` array containing the following dimensions of rectangles: `[[3, 4], [5, 12], [8, 15]]`.

1. We start by initializing `ans = 0` (to keep track of the area with the maximum diagonal) and `mx = 0` (to keep track of the square of the longest diagonal).
2. Let's begin the loop:
   - For the first rectangle `[3, 4]`, compute `t = 3**2 + 4**2 = 9 + 16 = 25`.
   - Compare `t` (25) with `mx` (0). Since 25 is greater than 0, update `mx = 25` and `ans = 3 * 4 = 12`.
3. Proceed to the second rectangle `[5, 12]`:
   - Compute `t = 5**2 + 12**2 = 25 + 144 = 169`.
   - Compare `t` (169) with `mx` (25). Since 169 is greater, update `mx = 169` and `ans = 5 * 12 = 60`.
4. Move to the third rectangle `[8, 15]`:
   - Compute `t = 8**2 + 15**2 = 64 + 225 = 289`.
   - Compare `t` (289) with `mx` (169). Since 289 is greater, update `mx = 289` and `ans = 8 * 15 = 120`.
5. The loop has finished. The final values are `mx = 289` and `ans = 120`. Thus, the maximum area of the rectangle with the longest diagonal is `120`.

In this example walkthrough, we see how the algorithm successfully identifies the rectangle with the longest diagonal and, consequently, the largest area among the rectangles with such a diagonal.

Solution Implementation

Python

```python
from typing import List

class Solution:
    def areaOfMaxDiagonal(self, dimensions: List[List[int]]) -> int:
        # Initialize the maximum diagonal length and area seen so far
        max_diagonal_length = 0
        max_area = 0

        # Loop through each dimension pair in the dimensions list
        for length, width in dimensions:
            # Calculate the diagonal length's square based on Pythagorean theorem
            diagonal_length_sq = length**2 + width**2

            # If a new maximum diagonal length is found
            if diagonal_length_sq > max_diagonal_length:
                # Update maximum diagonal length and area
                max_diagonal_length = diagonal_length_sq
                max_area = length * width
            # If current diagonal is equal to the maximum found but the area is larger
            elif diagonal_length_sq == max_diagonal_length:
                # Update the area to the larger one
                max_area = max(max_area, length * width)

        # Return the maximum area corresponding to the largest diagonal length
        return max_area
```

Java

```java
class Solution {
    public int areaOfMaxDiagonal(int[][] dimensions) {
        int maxArea = 0; // Holds the area of rectangle with largest diagonal so far
        int maxDiagonalSquare = 0; // Square of the largest diagonal seen so far

        // Loop through each dimensions array, where 'dimension' represents [length, width]
        for (int[] dimension : dimensions) {
            int length = dimension[0]; // Length of current rectangle
            int width = dimension[1];  // Width of current rectangle

            // Calculate the square of the diagonal for current rectangle
            int diagonalSquare = length * length + width * width;

            // If a larger diagonal is found, update maxDiagonalSquare and maxArea
            if (maxDiagonalSquare < diagonalSquare) {
                maxDiagonalSquare = diagonalSquare;
                maxArea = length * width; // Update area to the area of the current rectangle
            } else if (maxDiagonalSquare == diagonalSquare) {
                // If the same diagonal is found, update maxArea if current area is larger
                maxArea = Math.max(maxArea, length * width);
            }
        }

        // Return the area of the rectangle with the largest diagonal encountered
        return maxArea;
    }
}
```

C++

```cpp
class Solution {
public:
    // Function to calculate the area of the rectanale with the maximum diagonal length
    int areaOfMaxDiagonal(vector<vector<int>>& dimensions) {
        int maxArea = 0;        // Variable to store the maximum area found so far
        int maxDiagonalSq = 0; // Variable to store the square of the maximum diagonal length

        // Loop through the list of dimensions
        for (auto& dimension : dimensions) {
            int length = dimension[0];  // Store the length of the current rectangle
            int width = dimension[1];   // Store the width of the current rectanale
            int diagonalSq = length * length + width * width; // Calculate the square of the diagonal length

            // If the current diagonal is areater than the maximum found so far,
            // update the maximum diagonal and the maximum area accordingly
            if (maxDiagonalSq < diagonalSq) {
                maxDiagonalSq = diagonalSq;
                maxArea = length * width;
            // If the current diagonal equals the maximum diagonal found so far,
            // update the maximum area if the current area is greater
            } else if (maxDiagonalSq == diagonalSq) {
                maxArea = max(maxArea, length * width);
            }
        }

        // Return the maximum area found
        return maxArea;
    }
};
```

TypeScript

```typescript
function areaOfLargestDiagonal(dimensions: number[][]): number {
    let maxArea = 0; // Initialize maxArea to store the largest area found
    let maxDiagonalSquared = 0; // Initialize maxDiagonalSquared to store the square of the longest diagonal found

    // Iterate through each dimension pair in the dimensions array
    for (const [length, width] of dimensions) {
        const diagonalSquared = length * length + width * width; // Calculate the square of the diagonal for current dimensions

        // Compare the squared diagonal length with the maximum found so far
        if (maxDiagonalSquared < diagonalSquared) {
            maxDiagonalSquared = diagonalSquared; // Update maxDiagonalSquared
            maxArea = length * width; // Update maxArea with the area of the current dimensions
        } else if (maxDiagonalSquared === diagonalSquared) {
            maxArea = Math.max(maxArea, length * width); // If diagonals are equal, choose the larger area
        }
    }

    // Return the area of the rectangle with the longest diagonal
    return maxArea;
}

from typing import List

class Solution:
    def areaOfMaxDiagonal(self, dimensions: List[List[int]]) -> int:
        # Initialize the maximum diagonal length and area seen so far
        max_diagonal_length = 0
        max_area = 0

        # Loop through each dimension pair in the dimensions list
        for length, width in dimensions:
            # Calculate the diagonal length's square based on Pythagorean theorem
            diagonal_length_sq = length**2 + width**2

            # If a new maximum diagonal length is found
            if diagonal_length_sq > max_diagonal_length:
                # Update maximum diagonal length and area
                max_diagonal_length = diagonal_length_sq
                max_area = length * width
            # If current diagonal is equal to the maximum found but the area is larger
            elif diagonal_length_sq == max_diagonal_length:
                # Update the area to the larger one
                max_area = max(max_area, length * width)

        # Return the maximum area corresponding to the largest diagonal length
        return max_area
```

Time and Space Complexity

Time Complexity

The provided function `areaOfMaxDiagonal` processes each pair of dimensions in the input list exactly once. In each iteration, the function calculates the square of the length and width (`l**2 + w**2`) and compares it to the maximum diagonal length found so far (`mx < t` and `mx == t`). The calculations and comparisons inside the loop are constant time operations.

Let `n` be the number of pairs in the input list `dimensions`. Since the loop iterates over all pairs, the time complexity is O(n), where n is the length of the input list.

Space Complexity

The function uses a fixed number of variables (`ans`, `mx`, `t`, `l`, `w`) that do not depend on the size of the input list. Therefore, the space complexity is O(1), meaning it requires a constant amount of additional space.