Perfect Rectangle - Detailed Explanation

Table of Contents

Problem Statement

The problem requires us to determine if a set of axis-aligned rectangles can perfectly cover a single rectangular region without any gaps or overlaps. Each rectangle is represented as a list of four coordinates [x1, y1, x2, y2], which define its bottom-left and top-right corners. Our goal is to check if the given rectangles combine to form an exact rectangular shape without leaving any uncovered spaces or overlapping areas.

Intuition

To form a perfect rectangle, the total area covered by the given smaller rectangles must match the area of the bounding rectangle that encloses them. Additionally, all internal edges must be shared exactly twice, meaning they cancel each other out, while only four corner points of the final bounding rectangle should remain unique. This ensures that there are no extra overlapping regions or gaps.

Key Observations

Each rectangle has four corners, and when multiple rectangles are combined, internal corners should appear an even number of times, while the four extreme corners of the bounding rectangle should appear exactly once. This allows us to track unique points efficiently. Additionally, the sum of individual rectangle areas must be equal to the area of the expected bounding rectangle to confirm a perfect cover.

Approach

The solution involves four key steps. First, we determine the minimum and maximum x and y coordinates to identify the bounding rectangle. Second, we use a set to track unique corners of each rectangle, adding a corner if it appears once and removing it if it appears twice. By the end, only the four corners of the bounding rectangle should remain. Third, we calculate the total area covered by all rectangles and ensure it matches the computed area of the bounding rectangle. Finally, we validate that exactly four unique corners remain, corresponding to the expected extreme points of the bounding rectangle. If any of these conditions fail, we return False.

Edge Cases

There are several cases to consider. If there is a gap between rectangles, the algorithm must detect that not all areas are covered. If any rectangles overlap, it should also return False. A single rectangle case should return True since it forms a perfect cover by itself. Additionally, cases where rectangles extend beyond the expected bounding rectangle or create unintended smaller gaps need to be handled correctly.

Complexity Analysis

Time Complexity

The approach iterates through all given rectangles once to compute the area and track corners, resulting in an **O(n)** time complexity.

Space Complexity

A set is used to store unique corner points, which can have at most 4n entries in the worst case, leading to an **O(n)** space complexity.

Alternative Approaches

Instead of using a set, a hashmap could be used to count occurrences of each corner point, ensuring internal points appear an even number of times while extreme corners appear exactly once. Another alternative is a brute-force grid-based approach, where a 2D array represents the space covered by the rectangles, checking for overlaps or gaps. However, this method is inefficient for large inputs and results in an **O(W × H)** complexity, making it impractical.

Test Cases

The approach should be tested on different scenarios. A case with perfectly aligned rectangles should return True, while a case with a missing section should return False. Similarly, overlapping rectangles should result in False, and a single large rectangle should return True. Edge cases where rectangles extend beyond expected bounds should also be tested.

Final Thoughts

This problem effectively demonstrates the power of set operations and area calculations in efficiently determining whether a set of rectangles forms a perfect cover. By leveraging mathematical properties and optimal data structures, the solution achieves an **O(n) time and O(n) space complexity**, making it efficient even for large inputs. The key takeaway is that tracking corner points rather than checking all individual points is the most efficient way to solve this problem.