1037. Valid Boomerang

Easy Geometry Array Math

Problem Description

Given an array `points`, which contains three elements, where each element `points[i] = [xi, yi]` represents a coordinate on the X-Y plane, the task is to determine if these three points constitute a boomerang. A boomerang is defined as a set of three points that comply with two conditions: first, each point must be distinct from the others; and second, the points must not lie in a straight line — that is, they shouldn't all be collinear. The function should return `true` if the points form a boomerang, and `false` otherwise.

Intuition

To determine whether three points (`p1`, `p2`, and `p3`) form a boomerang, we need to ensure they are not collinear. A straightforward way to verify this is by checking if the slope between `p1` and `p2` is different from the slope between `p2` and `p3`. If both slopes are equal, the points lie on a straight line, which disqualifies them from forming a boomerang.

Mathematically, the slope between two points `(x1, y1)` and `(x2, y2)` is given by `(y2 - y1) / (x2 - x1)`. For points not to be collinear, the slopes `(y2 - y1) / (x2 - x1)` and `(y3 - y2) / (x3 - x2)` should be different. To avoid division by zero, we can cross-multiply and compare the products: `(y2 - y1) * (x3 - x2)` should not be equal to `(y3 - y2) * (x2 - x1)`.

The solution code implements this concept by taking the three points from the `points` array and calculating the products of differences as described, returning `true` if they're not equal and `false` otherwise. By cross-multiplying, we avoid the complication of dealing with the exact slope values or the divisions, simplifying our implementation and ensuring it remains robust even when vertical lines are involved (where the slope would be undefined).

The solution to this problem involves using the formula for the slope of a line and checking if the slope of the line between the first two points `(x1, y1)` and `(x2, y2)` is different from the slope of the line between the second two points `(x2, y2)` and `(x3, y3)`.

To avoid the division operation and potential division by zero errors when calculating the slope, the implementation uses cross multiplication.

Here is the algorithm in a step-by-step fashion:

1. Extract the coordinates of the three points from the input list.
2. Compute the product of differences for the first and second points: `(y2 - y1) * (x3 - x2)`.
3. Compute the product of differences for the second and third points: `(y3 - y2) * (x2 - x1)`.
4. Compare the two computed products. If they are equal, it indicates that the slopes are the same and hence the points are collinear. If the products are not equal, the points are not collinear.
5. Return `true` if the products are not equal (not collinear), else return `false`.

The above steps are represented in the given solution code:

```python
class Solution:
    def isBoomerang(self, points: List[List[int]]) -> bool:
        (x1, y1), (x2, y2), (x3, y3) = points
        return (y2 - y1) * (x3 - x2) != (y3 - y2) * (x2 - x1)
```

This code makes use of basic arithmetic operations and no additional data structures or complex patterns. It relies on the fact that if the product of the differences is equal for both pairs of points, then the three points lie on the same line (are collinear), which means they cannot form a boomerang. Otherwise, if the products are different, the points form a vertex of a non-straight line and hence do form a boomerang.

Example Walkthrough

Let's consider an example where we have three points `p1`, `p2`, and `p3` given by their coordinates: `p1 = [1, 1]`, `p2 = [2, 3]`, and `p3 = [3, 2]`. We want to find out if these points form a boomerang.

Using the provided solution approach, we will take the following steps:

1. Extract the coordinates of the three points from the input list. We already have that as:
   - `p1` (x1, y1) = (1, 1)
   - `p2` (x2, y2) = (2, 3)
   - `p3` (x3, y3) = (3, 2)
2. Compute the product of differences for the first and second points: `(y2 - y1) * (x3 - x2)`, which will be:
   - `(3 - 1) * (3 - 2) = 2 * 1 = 2`
3. Compute the product of differences for the second and third points: `(y3 - y2) * (x2 - x1)`, which will be:
   - `(2 - 3) * (2 - 1) = -1 * 1 = -1`
4. Compare the two computed products. In our example, `2` is not equal to `-1`, indicating that the slopes are different, and hence the points are not collinear.
5. Since the products are not equal, we return `true`, concluding that the points `p1`, `p2`, and `p3` do indeed form a boomerang.

To summarize, the points (1, 1), (2, 3), and (3, 2) when plugged into our solution approach show that they are not collinear and hence form a boomerang. The function will return `true` in this case.

Solution Implementation

Python

```python
from typing import List

class Solution:
    def isBoomerang(self, points: List[List[int]]) -> bool:
        # Extract the individual points for clarity
        point1, point2, point3 = points

        # Destructure the points into their respective x and y coordinates
        x1, y1 = point1
        x2, y2 = point2
        x3, y3 = point3

        # A boomerang is defined as a set of three points that are not in a straight line.
        # To determine if points are not in a straight line, the slope between points 1 and 2
        # must be different from the slope between points 2 and 3.

        # Calculate the slope between point1 and point2 (slope = (y2-y1)/(x2-x1))
        # Calculate the slope between point2 and point3 (slope = (y3-y2)/(x3-x2))
        # To avoid division by zero in slope calculations, compare the cross multiplication of
        # the differences in y-coordinates and x-coordinates instead.

        # A boomerang should meet the condition:
        # (y2 - y1)/(x2 - x1) != (y3 - y2)/(x3 - x2)
        # which simplifies to avoiding floating point precision comparison as:
        # (y2 - y1) * (x3 - x2) != (y3 - y2) * (x2 - x1)

        # Check if the slopes are different, if so, it is a boomerang
        return (y2 - y1) * (x3 - x2) != (y3 - y2) * (x2 - x1)

# Example usage:
# sol = Solution()
# print(sol.isBoomerang([[1,1], [2,3], [3,2]])) # Outputs: True, since the points form a boomerang
```

Java

```java
class Solution {
    public boolean isBoomerang(int[][] points) {
        // Extracting coordinates for better readability
        int x1 = points[0][0], y1 = points[0][1];
        int x2 = points[1][0], y2 = points[1][1];
        int x3 = points[2][0], y3 = points[2][1];

        // A boomerang is a set of three points that are all distinct and not in a straight line.
        // Check if the slopes between points p1 & p2 and points p2 & p3 are different.
        // Slope of line p1 and p2 is (y2 - y1)/(x2 - x1) and slope of line p2 and p3 is (y3 - y2)/(x3 - x2).
        // To avoid division (which can lead to division by zero) we cross-multiply to compare the slopes.
        return (y2 - y1) * (x3 - x2) != (y3 - y2) * (x2 - x1);
    }
}
```

C++

```cpp
#include <vector> // Include necessary header for the use of vector

class Solution {
public:
    bool isBoomerang(std::vector<std::vector<int>>& points) {
        // Extract coordinates of the first point
        int x1 = points[0][0];
        int y1 = points[0][1];

        // Extract coordinates of the second point
        int x2 = points[1][0];
        int y2 = points[1][1];

        // Extract coordinates of the third point
        int x3 = points[2][0];
        int y3 = points[2][1];

        // Check if the slope of the line formed by point 1 and point 2 is different
        // from the slope of the line formed by point 2 and point 3.
        // If slopes are different, points are non-collinear, thus returning true.
        return (y2 - y1) * (x3 - x2) != (y3 - y2) * (x2 - x1);
    }
};
```

TypeScript

```typescript
// This function checks if three points form a boomerang (a set of three points that are all distinct from each
// other and do not lie on the same line).
function isBoomerang(points: number[][]): boolean {
    // Destructuring the first point into x1 and y1
    const [x1, y1] = points[0];
    // Destructuring the second point into x2 and y2
    const [x2, y2] = points[1];
    // Destructuring the third point into x3 and y3
    const [x3, y3] = points[2];

    // Compute the slopes of the lines (x1,y1) -> (x2,y2) and (x2,y2) -> (x3,y3)
    // If the slopes are not equal, the points are non-collinear which means they form a boomerang.
    // To avoid division (and possible division by zero), cross-multiplication is used to compare the slopes:
    // slope of (x1,y1) -> (x2,y2) is (y2-y1)/(x2-x1)
    // slope of (x2,y2) -> (x3,y3) is (y3-y2)/(x3-x2)
    // We compare (y2-y1)*(x3-x2) with (y3-y2)*(x2-x1)
    return (x1 - x2) * (y2 - y3) !== (x2 - x3) * (y1 - y2);
}
```


```python
from typing import List

class Solution:
    def isBoomerang(self, points: List[List[int]]) -> bool:
        # Extract the individual points for clarity
        point1, point2, point3 = points

        # Destructure the points into their respective x and y coordinates
        x1, y1 = point1
        x2, y2 = point2
        x3, y3 = point3

        # A boomerang is defined as a set of three points that are not in a straight line.
        # To determine if points are not in a straight line, the slope between points 1 and 2
        # must be different from the slope between points 2 and 3.

        # Calculate the slope between point1 and point2 (slope = (y2-y1)/(x2-x1))
        # Calculate the slope between point2 and point3 (slope = (y3-y2)/(x3-x2))
        # To avoid division by zero in slope calculations, compare the cross multiplication of
        # the differences in y-coordinates and x-coordinates instead.

        # A boomerang should meet the condition:
        # (y2 - y1)/(x2 - x1) != (y3 - y2)/(x3 - x2)
        # which simplifies to avoiding floating point precision comparison as:
        # (y2 - y1) * (x3 - x2) != (y3 - y2) * (x2 - x1)

        # Check if the slopes are different, if so, it is a boomerang
        return (y2 - y1) * (x3 - x2) != (y3 - y2) * (x2 - x1)

# Example usage:
# sol = Solution()
# print(sol.isBoomerang([[1,1], [2,3], [3,2]])) # Outputs: True, since the points form a boomerang
```


Time and Space Complexity

The time complexity of the given code is `O(1)` because the operations performed are constant and do not depend on the size of the input; the code always handles exactly three points.

The space complexity of the code is `O(1)` as well, since the space used does not scale with the input. The only additional space used is for the unpacked point coordinates, which is a constant amount of space for the three pairs of integers.