1232. Check If It Is a Straight Line

Easy Geometry Array Math

Leetcode Link

Problem Description

The problem presents us with an array called `coordinates`. Each element within this array is itself an array that represents a point in the form `[x, y]`—where `x` is the x-coordinate and `y` is the y-coordinate of a point on the XY plane. The task is to determine if all these points lie on a single straight line or not.

Intuition

To check if points lie on the same straight line, we need to ensure that the slope between any two points is the same across all points. The slope is a measure of how steep a line is. If pairs of points have different slopes, the points do not lie on a single straight line.

To calculate the slope between two points `(x1, y1)` and `(x2, y2)`, we use the formula `slope = (y2 - y1) / (x2 - x1)`. If the slope is consistent between consecutive points, then all the points are on the same line.

In our solution, we take the first two points from `coordinates` to calculate a reference slope. For computational efficiency and to avoid the potential division by zero error, we check the equality of slopes using a cross-multiplication method. Specifically, instead of directly computing `(y2 - y1) / (x2 - x1) == (y - y1) / (x - x1)`, we check if `(x - x1) * (y2 - y1) == (y - y1) * (x2 - x1)` for subsequent points `(x, y)`.

If the equality holds for all pairs of points in the coordinates array, the function returns `True`, which means all points lie on a single straight line. If any pair of points fails the equality test, the function returns `False`, and we know that not all points are on the same line.

The intuition behind this approach is that we're using the properties of slopes in a way that's computationally efficient and avoids possible arithmetic errors when dealing with real numbers.

Solution Approach

The algorithm follows a straightforward approach by checking if the slope between any two points is the same across all points. No additional data structures or complex patterns are necessary. The algorithm involves simple arithmetic operations and a for loop. Here's a step-by-step implementation break down:

1. The algorithm starts by storing the x and y coordinates of the first two points from the `coordinates` array in `x1, y1` and `x2, y2`. These two points are used to find the reference slope for the line.

2. The algorithm then iterates over the remaining points in `coordinates`, starting from the third point, using a `for x, y in coordinates[2:]:` loop.

3. Inside the loop, for each point `(x, y)`, the algorithm checks if the cross-product of the differences in coordinates between point `(x, y)` and the first point `(x1, y1)` equals the cross-product of the differences between the second point `(x2, y2)` and the first point.

   The check is made using the following condition:

   ```
   if (x - x1) * (y2 - y1) != (y - y1) * (x2 - x1):
       return False
   ```

   This expression is derived from the slope equality `slope1 = slope2`. Instead of dividing the differences, which can cause a division by zero, it multiplies the opposite sides and compares them for equality.

4. If the check fails for any point, the function immediately returns `False`, indicating that the points do not all lie on the same straight line.

5. If the loop completes without encountering any point that fails the slope check, the function returns `True`, signaling that all points do indeed lie on a straight line.

The time complexity of this algorithm is O(n), where n is the number of points, since we have to check the condition for each point once. The space complexity is O(1) since we are using a fixed amount of additional space regardless of the number of points.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach:

Suppose `coordinates = [[1, 2], [2, 3], [3, 4]]`. We are looking to confirm whether these points fall on a single straight line.

1. Based on our algorithm, we first take the coordinates of the first two points to calculate the reference slope. So from our `coordinates`, we have `x1, y1 = 1, 2` and `x2, y2 = 2, 3`.

2. Next, the algorithm will iterate over the remaining points in `coordinates`. Since we only have three points in this example, it will iterate over just one point: `[3, 4]`.

3. Now, the algorithm will check if the cross-product of differences with the new point `[3, 4]`, denoted as `(x, y)`, and the first point `(x1, y1)` equals the cross-product of the differences between the second point `(x2, y2)` and the first point. This equation looks like:

   ```
   if (3 - 1) * (3 - 2) != (4 - 2) * (2 - 1):
   ```

   Simplifying the expressions on each side of the inequality gives:

   ```
   if 2 * 1 != 2 * 1:
   ```

   In this case, both sides are equal, meaning that the slope between point `[1, 2]` and point `[3, 4]` is the same as the slope between point `[1, 2]` and point `[2, 3]`.

4. Since there are no more points to check, and the equality held true for the third point, the algorithm would return `True`.

5. This confirms that all the given points in the example line `[1, 2], [2, 3], [3, 4]` are on the same straight line.

If we had a point that did not satisfy the slope equality, for example `[4, 5]` replaced by `[4, 6]`, the algorithm would compare:

```
if (4 - 1) * (3 - 2) != (6 - 2) * (2 - 1):
```

which simplifies to:

```
if 3 != 4:
```

This inequality is true, indicating that the point `[4, 6]` does not lie on the same straight line as the others, so the algorithm would return `False`.

Python Solution

```python
from typing import List

class Solution:
    def checkStraightLine(self, coordinates: List[List[int]]) -> bool:
        # Extract the first 2 points
        first_point = coordinates[0]
        second_point = coordinates[1]

        # Coordinates of the first point
        x1, y1 = first_point

        # Coordinates of the second point
        x2, y2 = second_point

        # Check if the subsequent points are in a straight line
        for point in coordinates[2:]:
            x, y = point

            # Use the cross product to determine if three points are on the same line:
            # (x - x1)/(x2 - x1) should be equal to (y - y1)/(y2 - y1)
            # Cross-multiplying to avoid division (to handle the case when x2 - x1 is 0) we get:
            # (x - x1) * (y2 - y1) should be equal to (y - y1) * (x2 - x1)
            # If they are not equal for any point, the points do not form a straight line
            if (x - x1) * (y2 - y1) != (y - y1) * (x2 - x1):
                return False

        # If the loop completes without returning False, all points are in a straight line
        return True
```

Java Solution

```java
class Solution {

    /**
     * Checks if all the given points lie on a straight line.
     *
     * @param coordinates Array of point coordinates on a 2D plane.
     * @return true if all points lie on a single straight line, false otherwise.
     */
    public boolean checkStraightLine(int[][] coordinates) {
        // Coordinates of the first point
        int x1 = coordinates[0][0];
        int y1 = coordinates[0][1];

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

        // Loop over the rest of the points starting from the third one
        for (int i = 2; i < coordinates.length; i++) {
            // Coordinates of the current point
            int currentX = coordinates[i][0];
            int currentY = coordinates[i][1];

            // Check if the current point lies on the line formed by the first two points
            // This is done by using the cross product which should be zero for collinear points
            int deltaX1 = currentX - x1;
            int deltaY1 = y2 - y1;
            int deltaY2 = currentY - y1;
            int deltaX2 = x2 - x1;

            // If current point does not satisfy the line equation then return false
            if (deltaX1 * deltaY1 != deltaY2 * deltaX2) {
                return false;
            }
        }

        // If all the points satisfy the line equation then return true
        return true;
    }
}
```

C++ Solution

```cpp
#include<vector> // Needed to use std::vector
using std::vector;

class Solution {
public:
    // Function to check if all points lie on a straight line.
    bool checkStraightLine(vector<vector<int>>& coordinates) {
        // Extract the first point (x1,y1)
        int x1 = coordinates[0][0], y1 = coordinates[0][1];
        // Extract the second point (x2,y2)
        int x2 = coordinates[1][0], y2 = coordinates[1][1];

        // Loop through all the remaining points
        for (int i = 2; i < coordinates.size(); ++i) {
            // Extract the current point (x,y)
            int x = coordinates[i][0], y = coordinates[i][1];

            // Check if the cross product of the vectors is zero
            // This is a vector algebra way of checking colinearity:
            // (x-x1)/(x2-x1) should be equal to (y-y1)/(y2-y1) for all points on the line
            // By cross-multiplying to avoid division (and potential division by zero),
            // we get (x-x1)*(y2-y1) == (y-y1)*(x2-x1)
            if ((x - x1) * (y2 - y1) != (y - y1) * (x2 - x1)) {
                return false; // The current point doesn't lie on the straight line defined by the first two points
            }
        }
        // All points lie on the same straight line
        return true;
    }
};
```

Typescript Solution

```typescript
// Necessary import statement when working with arrays in TypeScript
import { Vector } from 'prelude-ts';

// Function to check if all points lie on a straight line.
function checkStraightLine(coordinates: number[][]): boolean {
    // Extract the first point (x1, y1)
    const x1 = coordinates[0][0];
    const y1 = coordinates[0][1];

    // Extract the second point (x2, y2)
    const x2 = coordinates[1][0];
    const y2 = coordinates[1][1];

    // Loop through all the remaining points
    for (let i = 2; i < coordinates.length; i++) {
        // Extract the current point (x, y)
        const x = coordinates[i][0];
        const y = coordinates[i][1];

        // Check if the cross product of the vectors is zero
        // This is a vector algebra way of checking collinearity:
        // (x-x1)/(x2-x1) should be equal to (y-y1)/(y2-y1) for all points on the line
        // By cross-multiplying to avoid division (and potential division by zero),
        // we get (x-x1) * (y2-y1) == (y-y1) * (x2-x1)
        if ((x - x1) * (y2 - y1) !== (y - y1) * (x2 - x1)) {
            return false; // The current point doesn't lie on the straight line defined by the first two points
        }
    }

    // All points lie on the same straight line
    return true;
}
```

Time and Space Complexity

Time Complexity

The time complexity of the function `checkStraightLine` is `O(n)`, where `n` is the number of coordinate points in the input list `coordinates`. This is because the function uses a single loop that iterates through the coordinates starting from the third element, and for each iteration, it performs a constant number of mathematical operations to check if the points lie on the same line. These operations do not depend on the size of the input other than the fact that they iterate once for each element beyond the first two.

Space Complexity

The space complexity of the function is `O(1)`. The function uses a fixed amount of extra space to store the variables `x1`, `y1`, `x2`, `y2`, `x`, and `y`. The amount of space used does not scale with the size of the input list, meaning that the space usage is constant.