356. Line Reflection

Medium | Array | Hash Table | Math | Leetcode Link

Problem Description

The problem deals with a set of points on a 2D plane and the goal is to determine if there is a line parallel to the y-axis that can reflect all points symmetrically. In essence, we have to find out whether we can draw a vertical line such that every point has a mirrored counterpart on the other side of the line. The set of points before and after reflecting them across this line should be identical, even if some points are repeated.

Intuition

To solve this problem, we must recognize that a reflection over a line parallel to the y-axis means that each point `(x, y)` will have a mirrored point `(x', y)` such that both points are equidistant from the line of reflection. This implies that the sum of the x-coordinates of the two points (x + x') will be equal to twice the x-coordinate of the reflection line.

We approach the solution by first finding the minimal (`min_x`) and maximal (`max_x`) x-coordinates among all given points. The line of reflection, if it exists, would be exactly halfway between these two values, so the sum `s` for the reflection condition would be `min_x + max_x`.

Next, we store all given points in a set for constant-time lookups. Then, for every point in our input, we check if its symmetric counterpart with respect to the potential line of reflection (`(s - x, y)`) exists in our set. If all points satisfy this condition, the points are symmetric with respect to a line, and the function should return `true`. Otherwise, no such line exists and we return `false`.

Solution Approach

To implement the solution, we use a set data structure to store the points for efficient querying, as well as simple variables to keep track of the minimum and maximum x-coordinates.

The solution is straightforward in terms of algorithms and does not require any complex data structure manipulation or intricate patterns. It essentially involves the following steps:

1. Initialize `min_x` to infinity and `max_x` to negative infinity. These will be used to track the minimum and maximum x-values of the given points.
2. Traverse all points in the input list.
   - For each point `(x, y)`, update `min_x` and `max_x` to keep track of the smallest and largest x-coordinates.
   - Add the point `(x, y)` to the set `point_set` for efficient lookups later.
3. Calculate the sum `s` which is the potential reflection line's x-coordinate times two (`s = min_x + max_x`). This is based on the principle that the line of reflection would be exactly in the middle of the leftmost and rightmost points.
4. Finally, confirm that each point `(x, y)` has its mirrored counterpart. This is done by checking if `(s - x, y)` is present in `point_set` for every `(x, y)` in the list of points.
   - If all points have their mirrored counterparts in the set, return `true`.
   - If there is even a single point without a mirrored counterpart, return `false`.

The use of a set is crucial here as it allows us to verify the existence of the mirrored pairs in constant time (`O(1)`), thus keeping the entire algorithm's time complexity to `O(n)`, where `n` is the number of points.

There are no particular algorithms used beyond basic iteration and set operations. Also, there are no complex logical patterns or mathematical formulas involved, just the straightforward application of the definition of a reflection over a vertical line.

Example Walkthrough

Let's consider an example with the following set of points on a 2D plane: `[(1, 1), (3, 1), (7, 1), (9, 1)]`. Our goal is to determine if there exists a line parallel to the y-axis that reflects all these points symmetrically.

1. First, we initialize `min_x` to infinity and `max_x` to negative infinity to keep track of the minimum and maximum x-coordinates of the points.

2. Now we traverse our points list:

   - For point `(1, 1)`, we update `min_x` to 1 (as 1 < infinity) and `max_x` to 1 (as 1 > -infinity).
   - For point `(3, 1)`, `min_x` remains 1 and `max_x` is updated to 3.
   - For point `(7, 1)`, `min_x` remains 1 and `max_x` is updated to 7.
   - For the last point `(9, 1)`, `min_x` remains 1 and `max_x` is updated to 9.
   - Throughout the iteration, we also add each point to a set `point_set` for efficient lookups.
3. After the first traversal, we have `min_x = 1` and `max_x = 9`. The sum `s` which represents twice the possible line of reflection's x-coordinate is `s = min_x + max_x = 1 + 9 = 10`.

4. For the symmetry check, we traverse the list again:

   - Point `(1, 1)` has a counterpart `(9, 1)` because `s - x = 10 - 1 = 9`, and `(9, 1)` is in `point_set`.
   - Point `(3, 1)` should be mirrored with point `(7, 1)` because `s - x = 10 - 3 = 7`, and `(7, 1)` is indeed in `point_set`.
   - Since points `(7, 1)` and `(9, 1)` have already been paired with their counterparts, the conditions are satisfied for all points.

All points have a mirrored counterpart with respect to the line whose x-coordinate is `s/2`, which in this case is `5`. Since each point `(x, y)` has a matching point `(s - x, y)` in the set, we can conclude that it's possible to draw a vertical line parallel to the y-axis at `x = 5` that reflects all the points symmetrically. Therefore, the function should return `true` for this example.

Python Solution

```python
class Solution:
    def isReflected(self, points: List[List[int]]) -> bool:
        # Initialize minimum and maximum X to positive and negative infinity respectively.
        min_x, max_x = float('inf'), float('-inf')
        point_set = set()  # Create a set to store unique points.

        # Iterate over all points to find the min and max X values and add points to the set.
        for x, y in points:
            min_x = min(min_x, x)
            max_x = max(max_x, x)
            point_set.add((x, y))

        # Calculate the sum of min and max X, which should be equal to twice the X value of the reflection line.
        reflection_sum = min_x + max_x

        # Check if for each point (x, y), the reflected point across the Y-axis
        # given by (reflection_sum - x, y) exists in the point set.
        # The reflection across the Y-axis is defined by the line X = (min_x + max_x) / 2.
        return all((reflection_sum - x, y) in point_set for x, y in points)

```

Java Solution

```java
class Solution {
    public boolean isReflected(int[][] points) {
        // Initialize the max and min X coordinates to extreme values.
        final int MAX_VALUE = Integer.MAX_VALUE;
        int minX = MAX_VALUE;
        int maxX = Integer.MIN_VALUE;

        // Using a set to store unique point representations.
        Set<List<Integer>> pointSet = new HashSet<>();

        // Iterate over all points to find the minX and maxX values,
        // and add each point to the set.
        for (int[] point : points) {
            minX = Math.min(minX, point[0]);
            maxX = Math.max(maxX, point[0]);
            pointSet.add(List.of(point[0], point[1]));
        }

        // Calculate the sum of minX and maxX which is twice the X coordinate
        // of the line of reflection.
        int sum = minX + maxX;

        // Check if each point has its reflected counterpart in the set.
        for (int[] point : points) {
            // If the reflected point is not found in the set, return false.
            if (!pointSet.contains(List.of(sum - point[0], point[1]))) {
                return false;
            }
        }

        // If all points have their reflected counterpart, return true.
        return true;
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <set>
#include <algorithm>
using std::vector;
using std::set;
using std::min;
using std::max;
using std::pair;

class Solution {
public:
    // Function to determine if a set of points reflects across a single vertical axis.
    bool isReflected(vector<vector<int>>& points) {

        // Constants to represent infinity and negative infinity.
        const int INF = 1 << 30;

        // Variables to store the minimum and maximum X-coordinates.
        int minX = INF, maxX = -INF;

        // A set to store unique points (pair of X and Y coordinates).
        set<pair<int, int>> pointSet;

        // Loop through all points to populate pointSet and find the minX and maxX.
        for (auto& point : points) {
            minX = min(minX, point[0]); // Update minX if current point's X is smaller.
            maxX = max(maxX, point[0]); // Update maxX if current point's X is larger.
            pointSet.insert({point[0], point[1]}); // Insert the point into the set.
        }

        // Sum of minX and maxX is the total which, when halved, gives the X-coordinate of the reflection axis.
        int sumOfMinAndMaxX = minX + maxX;

        // Iterate through all points to check if the reflected point exists.
        for (auto& point : points) {
            // Calculate the reflection of the current point across the reflection axis.
            int reflectedX = sumOfMinAndMaxX - point[0];

            // Check if the reflected point is not in the set. If it's not, the point array is not reflected.
            if (!pointSet.count({reflectedX, point[1]})) {
                return false;
            }
        }

        // Return true if all points have their reflected counterparts.
        return true;
    }
};

```

Typescript Solution

```typescript
// Importing arrays and sets from ECMAScript 6 (ES6) standard.
import { Set } from "typescript-collections";

// Function to determine if a set of points reflects across a single vertical axis.
function isReflected(points: number[][]): boolean {

    // Constants to represent infinity and negative infinity.
    const INF: number = 1 << 30;

    // Variables to store the minimum and maximum X-coordinates.
    let minX: number = INF;
    let maxX: number = -INF;

    // A set to store unique points (tuple of X and Y coordinates).
    let pointSet: Set<string> = new Set();

    // Loop through all points to populate pointSet and find the minX and maxX.
    for (const point of points) {
        minX = Math.min(minX, point[0]); // Update minX if current point's X is smaller.
        maxX = Math.max(maxX, point[0]); // Update maxX if current point's X is larger.

        // Insert the point into the set as a string to ensure uniqueness.
        pointSet.add(JSON.stringify({x: point[0], y: point[1]}));
    }

    // Sum of minX and maxX is the total, which, when halved, gives the X-coordinate of the reflection axis.
    const sumOfMinAndMaxX: number = minX + maxX;

    // Iterate through all points to check if the reflected point exists.
    for (const point of points) {
        // Calculate the reflection of the current point across the reflection axis.
        const reflectedX: number = sumOfMinAndMaxX - point[0];

        // Check if the reflected point is not in the set. If it's not, the point array is not reflected.
        if (!pointSet.contains(JSON.stringify({x: reflectedX, y: point[1]}))) {
            return false;
        }
    }

    // Return true if all points have their reflected counterparts.
    return true;
}

```

Time and Space Complexity

Time Complexity

The time complexity of the given code can be analyzed as follows:

- We iterate over all `n` points once to find the minimum (`min_x`) and maximum (`max_x`) x-coordinates, and to add each point to the `point_set`. This requires `O(n)` time.

- Then we check if each point has a reflected point in the `point_set` using the formula `(s - x, y)` where `s` is the sum of `min_x` and `max_x`. This step also iterates over all `n` points and each lookup in a set is expected to be `O(1)` on average, leading to another `O(n)` time operation.

Therefore, the overall time complexity is `O(n)` + `O(n)` = `O(n)` where `n` is the number of input points.

Space Complexity

The space complexity of the code can be considered as follows:

- We create a set, `point_set`, which in the worst case, will contain all input points if all points are unique. This leads to a space complexity of `O(n)`.

- No additional significant space is used in the algorithm.

Hence, the overall space complexity of the algorithm is `O(n)` where `n` is the number of input points.