812. Largest Triangle Area

Easy Geometry Array Math

Problem Description

The given problem provides an array of points representing coordinates on the X-Y plane. Each point is defined as a pair of integers, `[xi, yi]`, which represent its X and Y coordinates, respectively. The task is to calculate the area of the largest triangle that can be formed by any three different points from the array. The resulting area should be approximated to within `10^-5` of the correct value to be considered acceptable.

To solve this problem, we need to understand that a triangle can be formed by any three non-collinear points on a plane. Hence, we must consider all possible combinations of three points from our array and calculate the area of the triangle they form. This is essentially a combinatorial problem where we seek the maximum value from a set of results obtained from these combinations.

Intuition

The intuitive solution for this problem leverages the mathematical formula for the area of a triangle when given three points. The formula to calculate the area `T` of a triangle with vertices at coordinates `(x1, y1)`, `(x2, y2)`, and `(x3, y3)` is:

```
T = |(x1(y2−y3) + x2(y3−y1) + x3(y1−y2)) / 2|
```

However, the provided solution code uses the cross product method to find the area of a triangle formed by three points in 2D space. Remember, in a 2D plane, the area `A` of a triangle formed by points `(x1, y1)`, `(x2, y2)`, and `(x3, y3)` can also be computed using the following formula derived from the cross product:

```
A = |(x1(y2−y3) + x2(y3−y1) + x3(y1−y2)) / 2|
```

This is equivalent to:

```
A = |((x2−x1)*(y3−y1) − (x3−x1)*(y2−y1)) / 2|
```

The formula is based on the fact that the absolute value of the cross product of two vectors can be interpreted as the area of the parallelogram they span, and the area of the triangle is half of that parallelogram.

The code implements a brute-force approach by considering every possible triplet of points:

1. Three nested loops iterate through all combinations of three points. It computes vectors `u1, v1` and `u2, v2` that represent two sides of a potential triangle.
2. The cross product of these vectors is calculated, which is essentially the determinant of a 2×2 matrix, and it corresponds to twice the area of the triangle formed by the points.
3. This value is halved and the absolute value is taken to get the area of the triangle. If the calculated area is greater than a previously recorded maximum area (`ans`), it updates the `ans` variable.
4. After exhausting all possible combinations, the largest recorded triangle area `ans` is returned.

The approach is exhaustive and straightforward but can be inefficient for large input sizes because it considers all possible point triplets, resulting in a time complexity of O(n^3), where `n` is the number of points. Nevertheless, it ensures that the maximum area is found.

Solution Approach

The solution approach utilizes a straightforward brute-force algorithm to solve the problem. Here's an in-depth breakdown of the implementation steps and the rationale behind them.

- **Algorithm**: The algorithm loops through each possible combination of three distinct points in the input array to compute the area of the triangle formed by those points. It uses the cross product method to determine the area, which works effectively for this 2D geometric problem.
- **Data Structures**: No specific data structures are needed for this solution except for the input list of points and a variable to keep track of the maximum area found.
- **Patterns**: The pattern here is the use of multiple nested loops to generate all possible triplets of points from the list. This pattern is common in problems where all combinations of elements need to be considered.

Here's a step-by-step explanation of the solution:

1. **Initial Maximum Area**: A variable called `ans` is initialized to zero, which will store the maximum triangle area found during the iteration.
2. **Nested Loops**: The three nested `for` loops are used to iterate over all possible combinations of three points from the `points` array. These loops pick one point in each iteration, creating triplets such as `[points[i], points[j], points[k]]`.
3. **Vector Calculation**: Within the innermost loop, two vectors are determined that represent the sides of the triangle from the point `(x1, y1)` to `(x2, y2)` and from `(x1, y1)` to `(x3, y3)`.
4. **Cross Product**: The cross product of these two vectors is computed as:

   ```
   CP = (u1 * v2 − u2 * v1)
   ```

   We consider only the 'z' component, which in a 3D cross product represents the area of the parallelogram formed by two vectors in 2D space.
5. **Area Calculation**: The actual area of the triangle is found by halving and taking the absolute value of the cross product `CP`, which is the determinant of the 2×2 matrix formed by vectors `u1, v1` and `u2, v2`:

   ```
   t = abs(u1 * v2 − u2 * v1) / 2
   ```

6. **Updating Maximum Area**: If the computed triangle area `t` is greater than the current maximum `ans`, it is updated to `t`.
7. **Result**: Once all combinations have been considered, and the maximum area has been found, the variable `ans` holds the largest triangle area, which is then returned as the final result.

This elementary brute-force solution ensures that even though it might not be efficient for a large number of points (due to its O(n^3) time complexity), it will always yield the correct maximum area of a triangle that can be formed, satisfying the problem requirements.

Example Walkthrough

Let's use a small set of points to illustrate the solution approach provided:

Suppose we have the following array of points: `[[0,0], [0,4], [3,0]]`.

This array forms a right-angled triangle with the right angle at the origin `(0,0)`. The area of this triangle can be calculated using the cross product approach explained in the content.

1. **Initial Maximum Area**: We initialize a variable `ans` to zero. This will store the maximum triangle area we find.
2. **Nested Loops**:
   - The first loop starts with `i = 0`, with `points[i]` will be `(0, 0)`.
   - The second loop starts with `j = 1`, so `points[j]` will be `(0, 4)`.
   - The third loop starts with `k = 2`, so `points[k]` will be `(3, 0)`.
3. **Vector Calculation**: We calculate two vectors:
   - From `(0,0)` to `(0,4)` which is `(0 - 0, 4 - 0)` or `(0,4)`.
   - From `(0,0)` to `(3,0)` which is `(3 - 0, 0 - 0)` or `(3,0)`.
4. **Cross Product**:
   - We compute the cross product `CP` of these vectors, which will be `0 * 0 - 3 * 4`, resulting in `-12`.
5. **Area Calculation**:
   - The absolute value of the cross product is `|CP| = |-12| = 12`.
   - The area of the triangle would be half of this, `t = CP / 2 = 6`.
6. **Updating Maximum Area**:
   - Since `t` is 6 and `ans` is 0, we update `ans` to 6.

With only one combination of points, the final `ans` we get is the maximum area, which is 6 in this case. Hence, the largest triangle that can be formed by any three points from the given array has an area equal to `6`.

By using the above steps on our small set of points, the brute-force algorithm effectively determines that the right-angled triangle formed by these points produces the largest area among all possible triangles formed from these points. This example demonstrates the underlying mechanics of the solution approach, where the algorithm evaluates all combinations of three points to find the triangle with the maximum area.

Solution Implementation

Python

```python
class Solution:
    def largest_triangle_area(self, points: List[List[int]]) -> float:
        # Initialize the max area to 0
        max_area = 0.0
        # Iterate over all possible combinations of three points to form triangles
        for x1, y1 in points:
            for x2, y2 in points:
                for x3, y3 in points:
                    # Calculate vector (u1, v1) from point 1 to point 2
                    u1, v1 = x2 - x1, y2 - y1
                    # Calculate vector (u2, v2) from point 1 to point 3
                    u2, v2 = x3 - x1, y3 - y1
                    # Calculate the triangle's area using the absolute value of the determinant
                    # (cross product of two vectors) divided by 2
                    area = abs(u1 * v2 - u2 * v1) / 2
                    # Update the max area if the current area is larger
                    max_area = max(max_area, area)
        # Return the largest found area of a triangle
        return max_area
```

Java

```java
class Solution {
    public double largestTriangleArea(int[][] points) {
        double maxArea = 0; // Initialize maxArea as the largest triangle area found so far.

        // Iterate through all the points to get the first point (p1) of the triangle.
        for (int[] point1 : points) {
            int x1 = point1[0], y1 = point1[1];

            // Iterate through all the points to get the second point (p2) of the triangle.
            for (int[] point2 : points) {
                int x2 = point2[0], y2 = point2[1];

                // Iterate through all the points to get the third point (p3) of the triangle.
                for (int[] point3 : points) {
                    int x3 = point3[0], y3 = point3[1];

                    // Calculate the vectors from point1 to point2 and from point1 to point3
                    int vector1X = x2 - x1, vector1Y = y2 - y1;
                    int vector2X = x3 - x1, vector2Y = y3 - y1;

                    // Calculate the area of the triangle using the determinant [u1 v1]
                    //                                                          [u2 v2]
                    // which is |v2 - x1  v2 - v1|
                    //          |x3 - x1  v3 - v1| divided by 2. Note: this represents
                    // the absolute value of the cross product of vectors (point1 to point2)
                    // and (point1 to point3) divided by 2.
                    double area = Math.abs(vector1X * vector2Y - vector2X * vector1Y) / 2.0;

                    // Update maxArea if the calculated area is larger than the current maxArea.
                    maxArea = Math.max(maxArea, area);
                }
            }
        }
        // Return the largest area found.
        return maxArea;
    }
}
```

C++

```cpp
#include <vector>
#include <cmath>
#include <algorithm>

class Solution {
public:
    double largestTriangleArea(vector<vector<int>>& points) {
        double largestArea = 0; // Initialize the variable to hold the largest triangle area
        // Iterate over all possible combinations of three points to find the largest area
        for (auto& point1 : points) {
            int x1 = point1[0], y1 = point1[1];
            for (auto& point2 : points) {
                int x2 = point2[0], y2 = point2[1];
                for (auto& point3 : points) {
                    int x3 = point3[0], y3 = point3[1];
                    // Calculate the vector from point1 to point2 and point1 to point3
                    int vector1X = x2 - x1, vector1Y = y2 - y1;
                    int vector2X = x3 - x1, vector2Y = y3 - y1;
                    // Use the cross product of vectors to find the area of the triangle
                    double area = std::abs(vector1X * vector2Y - vector2X * vector1Y) / 2.0;
                    // Update largestArea if a larger area is found
                    largestArea = std::max(largestArea, area);
                }
            }
        }
        return largestArea; // Return the result
    }
};
```

TypeScript

```typescript
function largestTriangleArea(points: number[][]): number {
    let largestArea: number = 0; // Initialize the variable to hold the largest triangle area

    // Triple nested loop to iterate over all possible combinations of three points
    points.forEach(point1 => {
        const x1 = point1[0], y1 = point1[1];

        points.forEach(point2 => {
            const x2 = point2[0], y2 = point2[1];

            points.forEach(point3 => {
                const x3 = point3[0], y3 = point3[1];

                // Calculate the vectors from point1 to point2, and from point1 to point3
                const vector1X = x2 - x1, vector1Y = y2 - y1;
                const vector2X = x3 - x1, vector2Y = y3 - y1;

                // Calculate the cross product of the vectors to get twice the area of the triangle
                const area = Math.abs(vector1X * vector2Y - vector2X * vector1Y) / 2.0;

                // Update largestArea if a larger area is found
                largestArea = Math.max(largestArea, area);
            });
        });
    });

    // Return the largest area found
    return largestArea;
}

// Example usage:
const examplePoints: number[][] = [[0, 0], [0, 1], [1, 0], [0, 2], [2, 0]];
console.log(largestTriangleArea(examplePoints)); // Possible output: 2 (for the triangle formed by the last three points)
```

Rust

```rust
class Solution:
    def largest_triangle_area(self, points: List[List[int]]) -> float:
        # Initialize the max area to 0
        max_area = 0.0
        # Iterate over all possible combinations of three points to form triangles
        for x1, y1 in points:
            for x2, y2 in points:
                for x3, y3 in points:
                    # Calculate vector (u1, v1) from point 1 to point 2
                    u1, v1 = x2 - x1, y2 - y1
                    # Calculate vector (u2, v2) from point 1 to point 3
                    u2, v2 = x3 - x1, y3 - y1
                    # Calculate the triangle's area using the absolute value of the determinant
                    # (cross product of two vectors) divided by 2
                    area = abs(u1 * v2 - u2 * v1) / 2
                    # Update the max area if the current area is larger
                    max_area = max(max_area, area)
        # Return the largest found area of a triangle
        return max_area
```

Time and Space Complexity

The time complexity of the code is `O(n^3)` where `n` is the number of points provided in the `points` list. This is because there are three nested loops each iterating over all the points.

The space complexity of the code is `O(1)` because the only extra space used is for variables to store the current point coordinates, the components of two vectors (`u1`, `v1`, `u2`, `v2`), and the area of the current triangle (`t`). These variables require a constant amount of space regardless of the input size.