593. Valid Square

May 20, 2017 | 25.9K views

Previous Next

Given the coordinates of four points in 2D space, return whether the four points could construct a square.

The coordinate (x,y) of a point is represented by an integer array with two integers.

Example:

```
Input: p1 = [0,0], p2 = [1,1], p3 = [1,0], p4 = [0,1]
Output: True
```

Note:

1. All the input integers are in the range [-10000, 10000].
2. A valid square has four equal sides with positive length and four equal angles (90-degree angles).
3. Input points have no order.

Solution

Approach #1 Brute Force [Accepted]

The idea behind determining whether 4 given set of points constitute a valid square or not is really simple. Firstly, we need to determine if the sides of the quadrilateral formed by these 4 points are equal. But checking only this won't suffice. Since, this condition will be satisfied even in the case of a rhombus, where all the four sides are equal but the adjacent sides aren't perpendicular to each other. Thus, we also need to check if the lengths of the diagonals formed between the corners of the quadrilateral are equal. If both the conditions are satisfied, then only the given set of points can be deemed appropriate for constituting a square.

Now, the problem arises in determining which pairs of points act as the adjacent points on the square boundary. So, the simplest method is to consider every possible case. For the given 4 points, $[p_1, p_1, p_2, p_3]$, there are a total of 4! ways in which these points can be arranged to be considered as the square's boundaries. We can generate every possible permutation and check if any permutation leads to the valid square arrangement of points.

Java

```
public class Solution {
    public double dist(int[] p1, int[] p2) {
        return (p2[1] - p1[1]) * (p2[1] - p1[1]) + (p2[0] - p1[0]) * (p2[0] - p1[0]);
    }
    public boolean check(int[] p1, int[] p2, int[] p3, int[] p4) {
        return dist(p1,p2) > 0 && dist(p1, p2) == dist(p2, p3) && dist(p2, p3) == dist(p3, p4) &&
dist(p3, p4) == dist(p4, p1) && dist(p1, p3) == dist(p2, p4);
    }
    public boolean validSquare(int[] p1, int[] p2, int[] p3, int[] p4) {
        int[][] p = {p1,p2,p3,p4};
        return checkAllPermutations(p, 0);
    }
    boolean checkAllPermutations(int[][] p, int l) {
        if (l == 4) {
            return check(p[0], p[1], p[2], p[3]);
        } else {
            boolean res = false;
            for (int i = l; i < 4; i++) {
                swap(p, l, i);
                res |= checkAllPermutations(p, l + 1);
                swap(p, l, i);
            }
            return res;
        }
    }
    public void swap(int[][] p, int x, int y) {
        int[] temp = p[x];
        p[x] = p[y];
        p[y] = temp;
    }
}
```

Complexity Analysis

- Time complexity : $O(1)$. Constant number of permutations(4!) are generated.
- Space complexity : $O(1)$. Constant space is required.

Approach #2 Using Sorting [Accepted]

Instead of considering all the permutations of arrangements possible, we can make use of maths to simplify this problem a bit. If we sort the given set of points based on their x-coordinate values, and in the case of a tie, based on their y-coordinate value, we can obtain an arrangement, which directly reflects the arrangement of points on a valid square boundary possible.

Consider the only possible cases as shown in the figure below:

In each case, after sorting, we obtain the following conclusion regarding the connections of the points:

1. p_0p_1, p_1p_3, p_3p_2 and p_2p_0 form the four sides of any valid square.
2. p_0p_3 and p_1p_2 form the diagonals of the square.

Thus, once the sorting of the points is done, based on the above knowledge, we can directly compare p_0p_1, p_1p_3, p_3p_2 and p_2p_0 for equality of lengths(corresponding to the sides); and p_0p_3 and p_1p_2 for equality of lengths(corresponding to the diagonals).

Java

```
public class Solution {
    public double dist(int[] p1, int[] p2) {
        return (p2[1] - p1[1]) * (p2[1] - p1[1]) + (p2[0] - p1[0]) * (p2[0] - p1[0]);
    }
    public boolean validSquare(int[] p1, int[] p2, int[] p3, int[] p4) {
        int[][] p={p1,p2,p3,p4};
        Arrays.sort(p, (i1, i2) -> i1[0] == i2[0] ? i1[1] - i2[1] : i1[0] - i2[0]);
        return dist(p[0], p[1]) != 0 && dist(p[0], p[1]) == dist(p[1], p[3]) && dist(p[1], p[3]) ==
dist(p[3], p[2]) && dist(p[3], p[2]) == dist(p[2], p[0]) && dist(p[0],p[3])==dist(p[1],p[2]);
    }
}
```

Complexity Analysis

- Time complexity : $O(1)$. Sorting 4 points takes constant time.
- Space complexity : $O(1)$. Constant space is required.

Approach #3 Checking every case [Accepted]

Algorithm

If we consider all the permutations descripting the arrangement of points as in the brute force approach, we can come up with the following set of 24 arrangements:

Sr. No.	Order of points	Sides	Diagonals
1	$p_0p_1p_2p_3$	p_0p_1 p_1p_2 p_2p_3 p_3p_0	p_0p_2 p_1p_3
2	$p_0p_1p_3p_2$	p_0p_1 p_1p_3 p_3p_2 p_2p_0	p_0p_3 p_1p_2
3	$p_0p_2p_1p_3$	p_0p_2 p_2p_1 p_1p_3 p_3p_0	p_0p_1 p_2p_3
4	$p_0p_2p_3p_1$	p_0p_2 p_2p_3 p_3p_1 p_1p_0	p_0p_3 p_2p_1
5	$p_0p_3p_1p_2$	p_0p_3 p_3p_1 p_1p_2 p_2p_0	p_0p_1 p_3p_2
6	$p_0p_3p_2p_1$	p_0p_3 p_3p_2 p_2p_1 p_1p_0	p_0p_2 p_3p_1
7	$p_1p_0p_2p_3$	p_1p_0 p_0p_2 p_2p_3 p_3p_1	p_1p_2 p_0p_3
8	$p_1p_0p_3p_2$	p_1p_0 p_0p_3 p_3p_2 p_2p_1	p_1p_3 p_0p_2
9	$p_1p_2p_0p_3$	p_1p_2 p_2p_0 p_0p_3 p_3p_1	p_1p_0 p_2p_3
10	$p_1p_2p_3p_0$	p_1p_2 p_2p_3 p_3p_0 p_0p_1	p_1p_3 p_2p_0
11	$p_1p_3p_0p_2$	p_1p_3 p_3p_0 p_0p_2 p_2p_1	p_1p_0 p_3p_2
12	$p_1p_3p_2p_0$	p_1p_3 p_3p_2 p_2p_0 p_0p_1	p_1p_2 p_3p_0

Sr. No.	Order of points	Sides	Diagonals
13	$p_2p_0p_1p_3$	p_2p_0 p_0p_1 p_1p_3 p_3p_2	p_2p_1 p_0p_3
14	$p_2p_0p_3p_1$	p_2p_0 p_0p_3 p_3p_1 p_1p_2	p_2p_3 p_0p_1
15	$p_2p_1p_0p_3$	p_2p_1 p_1p_0 p_0p_3 p_3p_2	p_2p_0 p_1p_3
16	$p_2p_1p_3p_0$	p_2p_1 p_1p_3 p_3p_0 p_0p_2	p_2p_3 p_1p_0
17	$p_2p_3p_0p_1$	p_2p_3 p_3p_0 p_0p_1 p_1p_2	p_2p_0 p_3p_1
18	$p_2p_3p_1p_0$	p_2p_3 p_3p_1 p_1p_0 p_0p_2	p_2p_1 p_3p_0
19	$p_3p_0p_1p_2$	p_3p_0 p_0p_1 p_1p_2 p_2p_3	p_3p_1 p_0p_2
20	$p_3p_0p_2p_1$	p_3p_0 p_0p_2 p_2p_1 p_1p_3	p_3p_2 p_0p_1
21	$p_3p_1p_0p_2$	p_3p_1 p_1p_0 p_0p_2 p_2p_3	p_3p_0 p_1p_2
22	$p_3p_1p_2p_0$	p_3p_1 p_1p_2 p_2p_0 p_0p_3	p_3p_2 p_1p_0
23	$p_3p_2p_0p_1$	p_3p_2 p_2p_0 p_0p_1 p_1p_3	p_3p_0 p_2p_1
24	$p_3p_2p_1p_0$	p_3p_2 p_2p_1 p_1p_0 p_0p_3	p_3p_1 p_2p_0

In this figure, the rows with the same shaded color indicate that the corresponding arrangements lead to the same set of edges and diagonals. Thus, we can see that only three unique cases exist. Thus, instead of generating all the 24 permutations, we check for the equality of edges and diagonals for only the three distinct cases.

Java

```
public class Solution {
    public double dist(int[] p1, int[] p2) {
        return (p2[1] - p1[1]) * (p2[1] - p1[1]) + (p2[0] - p1[0]) * (p2[0] - p1[0]);
    }
    public boolean check(int[] p1, int[] p2, int[] p3, int[] p4) {
        return dist(p1,p2) > 0 && dist(p1, p2) == dist(p2, p3) && dist(p2, p3) == dist(p3, p4) &&
dist(p3, p4) == dist(p4, p1) && dist(p1, p3) == dist(p2, p4);
    }
    public boolean validSquare(int[] p1, int[] p2, int[] p3, int[] p4) {
        return check(p1, p2, p3, p4) || check(p1, p3, p2, p4) || check(p1, p2, p4, p3);
    }
}
```

Complexity Analysis

- Time complexity : $O(1)$. A fixed number of comparisons are done.
- Space complexity : $O(1)$. No extra space required.

Rate this article:

Previous Next

Comments: 16

Sort By

Type comment here... (Markdown is supported)

Preview Post

ziha03 ★ 38 November 29, 2019 8:53 AM

the simplest solution can get every distance between every two points and add to a set return set.size() == 2

a square can only have 2 different lengths between two points, Also, any 4 points satisfy conditions below must be a square

Read More

25 Share Reply

SHOW 4 REPLIES

WinterLi ★ 1 December 8, 2017 5:57 PM

What I did is check the vectors from the center to four points
while the length of any one of these vectors must be same and not zero
and angle among these four vectors must be 90-degree or 180-degree.

length_Sq > 0 // not a dot

Read More

6 Share Reply

dangal ★ 30 November 28, 2019 7:06 AM

Idea: In any square there will be 4 lines of same length and 2 lines of same length
Code: https://leetcode.com/problems/valid-square/discuss/437680/Square-Distance-and-HashTable-for-easy-calculation

2 Share Reply

SHOW 1 REPLY

Lucy18 ★ 2 August 10, 2017 12:15 PM

You don't consider the situation that a diamond but its angles are not 90-degree. (For example,it angles are 60-degree,120-degree,60-degree,120-degree like put two equilateral triangles together). I think your solution should add "the length of four sides != the length of diagonals" in order to avoid this case. And forgive my poor english...

1 Share Reply

Lucy18 ★ 2 August 10, 2017 12:23 PM Report

Vote Up

Understand that! My mistake.

0 Share Reply

haimei2 ★ 168 May 23, 2017 12:22 AM

I also consider using sort to solve the problem, not by sorting the four points position but sorting the length of two points. Finally, [a, a, a, a, 2a, 2a] is the correct pattern.

```
public class Solution {
    public boolean validSquare(int[] p1, int[] p2, int[] p3, int[] p4) {
```

Read More

0 Share Reply

vinod23 ★ 461 May 21, 2017 4:39 PM

@ManueP I have updated the statement. Hope it seems correct now.

0 Share Reply

user1434C ★ 0 July 8, 2020 11:02 AM

Regarding solution 2, we do not need to check the length of all edges. If we could check the diagonals are same and two of the other end edges are same, this should be enough.

Optimized solution below.

Read More

0 Share Reply

pgmreddy ★ 516 June 20, 2020 9:05 AM

Few questions for solution creator:

1. Should the solution tell that distance between two points is `sqrt( (x1-x2)^2 + (y1-y2)^2 )` but `sqrt` is not required to check here ?
2. Should the solution talk about the edge case of all points being (0,0) and even having

Read More

0 Share Reply

Mahr ★ 71 May 7, 2020 3:23 AM Report

I think it is better to remind in the problem description that we also need to check if a figure is a rhombus/diamond which is a special case of rectangle and is also considered as a square. Missed that and now I need to re-write my solution :(

0 Share Reply