Count maximum points on same line

Difficulty Level : Hard • Last Updated : 05 Jul, 2022

Given N point on a 2D plane as pair of (x, y) co-ordinates, we need to find maximum number of point which lie on the same line.

Examples:

```
Input : points[] = {-1, 1}, {0, 0}, {1, 1},
                   {2, 2}, {3, 3}, {3, 4}
Output : 4
Then maximum number of point which lie on same
line are 4, those point are {0, 0}, {1, 1}, {2, 2},
{3, 3}
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

We can solve above problem by following approach – For each point p, calculate its slope with other points and use a map to record how many points have same slope, by which we can find out how many points are on same line with p as their one point. For each point keep doing the same thing and update the maximum number of point count found so

1. if two point are (x1, y1) and (x2, y2) then their slope will be (y2 – y1) / (x2 – x1) which can be a double value and can cause precision problems. To get rid of the precision problems, we treat slope as pair ((y2 – y1), (x2 – x1)) instead of ratio and reduce pair by their gcd before inserting into map. In below code points which are vertical or repeated are treated separately.
2. If we use unordered_map in c++ or HashMap in Java for storing the slope pair, then total time complexity of solution will be O(n^2) and space complexity will be O(n).

Implementation:

C++

```
/* C/C++ program to find maximum number of point
which lie on same line */
#include <bits/stdc++.h>
#include <boost/functional/hash.hpp>

using namespace std;

// method to find maximum collinear point
int maxPointOnSameLine(vector< pair<int, int> > points)
{
    int N = points.size();
    if (N < 2)
        return N;

    int maxPoint = 0;
    int curMax, overlapPoints, verticalPoints;

    // here since we are using unordered_map
    // which is based on hash function
    //But by default we don't have hash function for pairs
    //so we'll use hash function defined in Boost library
    unordered_map<pair<int, int>, int,boost::
```

```
{
    curMax = overlapPoints = verticalPoints = 0;

    // looping from i + 1 to ignore same pair again
    for (int j = i + 1; j < N; j++)
    {
        // If both point are equal then just
        // increase overlapPoint count
        if (points[i] == points[j])
            overlapPoints++;

        // If x co-ordinate is same, then both
        // point are vertical to each other
        else if (points[i].first == points[j].first)
            verticalPoints++;

        else
        {
            int yDif = points[j].second - points[i].second;
            int xDif = points[j].first - points[i].first;
            int g = __gcd(xDif, yDif);

            // reducing the difference by their gcd
            yDif /= g;
            xDif /= g;

            // increasing the frequency of current slope
            // in map
            slopeMap[make_pair(yDif, xDif)]++;
            curMax = max(curMax, slopeMap[make_pair(yDif, xDif)]);
        }

        curMax = max(curMax, verticalPoints);
    }

    // updating global maximum by current point's maximum
    maxPoint = max(maxPoint, curMax + overlapPoints + 1);

    // printf("maximum collinear point
    // which contains current point
    // are : %d\n", curMax + overlapPoints + 1);
```

```
}

// Driver code
int main()
{
    const int N = 6;
    int arr[N][2] = {{-1, 1}, {0, 0}, {1, 1}, {2, 2},
                     {3, 3}, {3, 4}};

    vector< pair<int, int> > points;
    for (int i = 0; i < N; i++)
        points.push_back(make_pair(arr[i][0], arr[i][1]));

    cout << maxPointOnSameLine(points) << endl;

    return 0;
}
```

Python3

```
# python3 program to find maximum number of 2D points that lie on the sam

from collections import defaultdict
from math import gcd
from typing import DefaultDict, List, Tuple

IntPair = Tuple[int, int]


def normalized_slope(a: IntPair, b: IntPair) -> IntPair:
    """
    Returns normalized (rise, run) tuple. We won't return the actual rise
    result in order to avoid floating point math, which leads to faulty
    comparisons.

    See
    https://en.wikipedia.org/wiki/Floating-point_arithmetic#Accuracy_probl
    """
    run = b[0] - a[0]
```

```
    # normalize to left-to-right
    if run < 0:
        a, b = b, a
        run = b[0] - a[0]

    rise = b[1] - a[1]
    # Normalize by greatest common divisor.
    # math.gcd only works on positive numbers.
    gcd_ = gcd(abs(rise), run)
    return (
        rise // gcd_,
        run // gcd_,
    )


def maximum_points_on_same_line(points: List[List[int]]) -> int:
    # You need at least 3 points to potentially have non-collinear points
    # For [0, 2] points, all points are on the same line.
    if len(points) < 3:
        return len(points)

    # Note that every line we find will have at least 2 points.
    # There will be at least one line because len(points) >= 3.
    # Therefore, it's safe to initialize to 0.
    max_val = 0

    for a_index in range(0, len(points) - 1):
        # All lines in this iteration go through point a.
        # Note that lines a-b and a-c cannot be parallel.
        # Therefore, if lines a-b and a-c have the same slope, they're th
        # line.
        a = tuple(points[a_index])
        # Fresh lines already have a, so default=1
        slope_counts: DefaultDict[IntPair, int] = defaultdict(lambda: 1)

        for b_index in range(a_index + 1, len(points)):
            b = tuple(points[b_index])
            slope_counts[normalized_slope(a, b)] += 1

        max_val = max(
            max val.
```

```
print(maximum_points_on_same_line([
    [-1, 1],
    [0, 0],
    [1, 1],
    [2, 2],
    [3, 3],
    [3, 4],
]))

# This code is contributed by Jose Alvarado Torre
```

Javascript

```
/* JavaScript program to find maximum number of point
which lie on same line */

// Function to find gcd of two numbers
let gcd = function(a, b) {
  if (!b) {
    return a;
  }
  return gcd(b, a % b);
}

// method to find maximum collinear point
function maxPointOnSameLine(points){
    let N = points.length;
    if (N < 2){
        return N;
    }

    let maxPoint = 0;
    let curMax, overlapPoints, verticalPoints;

    // Creating a map for storing the data.
    let slopeMap = new Map();

    // looping for each point
```

```
        verticalPoints = 0;

        // looping from i + 1 to ignore same pair again
        for (let j = i + 1; j < N; j++)
        {
            // If both point are equal then just
            // increase overlapPoint count
            if (points[i] === points[j]){
                overlapPoints++;
            }

            // If x co-ordinate is same, then both
            // point are vertical to each other
            else if (points[i][0] === points[j][0]){
                verticalPoints++;
            }
            else{
                let yDif = points[j][1] - points[i][1];
                let xDif = points[j][0] - points[i][0];
                let g = gcd(xDif, yDif);

                // reducing the difference by their gcd
                yDif = Math.floor(yDif/g);
                xDif = Math.floor(xDif/g);

                // increasing the frequency of current slope.
                let tmp = [yDif, xDif];
                if(slopeMap.has(tmp.join(''))){
                    slopeMap.set(tmp.join(''), slopeMap.get(tmp.join(''))
                }
                else{
                    slopeMap.set(tmp.join(''), 1);
                }

                curMax = Math.max(curMax, slopeMap.get(tmp.join('')));
            }

            curMax = Math.max(curMax, verticalPoints);
        }

        // updating global maximum by current point's maximum
```

```
        // are : %d\n", curMax + overlapPoints + 1);
        slopeMap.clear();
    }

    return maxPoint;
}

// Driver code
{
    let N = 6;
    let arr = [[-1, 1], [0, 0], [1, 1], [2, 2],
                    [3, 3], [3, 4]];

    console.log(maxPointOnSameLine(arr));
}

// The code is contributed by Gautam goel (gautamgoel962)
```

Output

```
4
```

Time Complexity: O(n^{2}logn), where n denoting length of string.
Auxiliary Space: O(n).

This article is contributed by **Utkarsh Trivedi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Liked 49

Previous

Find the Missing Point of Parallelogram

Next

How to check if a given point lies inside or outside a polygon?

RECOMMENDED ARTICLES

01 **Check whether two points (x1, y1) and (x2, y2) lie on same side of a given line or not**
13, Aug 19

02 **Minimize the maximum distance between adjacent points after adding K points anywhere in between**
27, Sep 21

03 **Count of obtuse angles in a circle with 'k' equidistant points between 2 given points**

05 **Count pairs of same parity indexed elements with same MSD after replacing each element by the sum of maximum digit * A and minimum digits * B**
08, Mar 21

06 **Prime points (Points that split a number into two primes)**
24, Mar 17

07 **Ways to choose three points with distance between the most distant points <= L**
31, May 18

the number line
02, Apr 19

points
01, Nov 18

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

Improved By : PortgasDAce, RahulSingal, sagar0719kumar, josalvatorre, sweetyty, gautamgoel962, shivamanandrj9, hardikkoriintern

Article Tags : Amazon, Geometric, Hash, Mathematical

Practice Tags : Amazon, Hash, Mathematical, Geometric

Improve Article | Report Issue

feedback@geeksforgeeks.org