149. Max points on a line

June 25, 2020 | 3.6K views

Average Rating: 3.40 (15 votes)

Given *n* points on a 2D plane, find the maximum number of points that lie on the same straight line.

Example 1:

```
Input: [[1,1],[2,2],[3,3]]
Output: 3
Explanation:
^
|
|        o
|     o
|  o
+------------>
0  1  2  3  4
```

Example 2:

```
Input: [[1,1],[3,2],[5,3],[4,1],[2,3],[1,4]]
Output: 4
Explanation:
^
|
|  o
|     o        o
|        o
|  o        o
+----------------------->
0  1  2  3  4  5  6
```

NOTE: input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

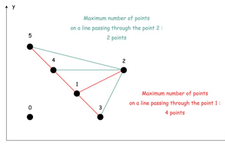
Solution

Approach 1: Enumeration

Intuition

Let's simplify the problem and search the maximum number of points on a line passing through the point `i`.

One could immediately notice that it's interesting to consider only the next points `i + 1 .. N - 1` because the maximum number of points containing, for example, the point `i - 2` was already found during the search of the maximum number of points on a line passing through the point `i - 2`.



The idea is very simple : draw the lines passing through the point `i` and each of the next points. Save these lines is a hash table with a counter `2` = two points on this line.
Let's imagine now that points `i < i + k < i + 1` are on the same line. Then drawing a line through `i` and `i + 1` one would discover that this line is already tracked and hence one has to update a counter of points on this line `count++`.

> How to save a line?

If the line is horizontal, i.e. `y = c`, one could use this constant `c` as a line key in a hash table of horizontal lines.

The other lines could be represented as `y = slope * x + c`.

The equation for the line passing through two points `1` and `2` could be written through their coordinates as

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}$$

that for the representation $y = \text{slope} \times x + c$ means

$$\text{slope} = \frac{y_2 - y_1}{x_2 - x_1}$$

> Since we are drawing a line between the starting point `i` and each of the following points, if all these lines share the same `slope` value, then we can be sure that all these points are aligned on the same line.

Hence, a `slope` value is sufficient to represent a unique line starting from a specific point `i`.

One might go ahead and use a float (or double) value to represent each unique slope. Indeed, this could work for most of the cases, but not all.

Slope Representation

One of the cases that a float value would not cut for the slope variable is that when two points form a vertical line, (i.e. $x_1 == x_2$). As we can see from the formula to calculate the slope value, we would encounter a divide-by-zero error.

One might argument we could treat this as a special case, and assign a special value (say, zero) to represent the horizontal slope.

However, a bigger problem is that the float and double values are intrinsically ***inaccurate***, due to how these values are represented in the computer. A simple fact to comprehend this limitation is that we could have infinite number of digits for a fraction number (i.e. $\frac{1}{3}$), we could only keep a limited number of digits as its float value in the computer.

One can run a fun experiment to calculate the result for the operation of $1.2 - 1.0$ in the Python shell. (*spoiler alert*: we would get the value of `0.19999999999999996` as the result.)

Therefore, it is not wise to use the float/double value to represent a unique slope, since they are not accurate.

> To circumvent the above issue, one could use a pair of co-prime integers to represent unique slope.

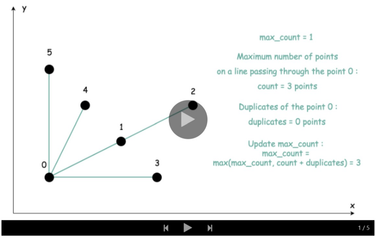
As a reminder, two integers are co-primes, if and only if their greatest common divisor is 1.

As one can see, due to the property of co-prime numbers, they can be used to represent the slope values of different lines. For example, for the slope values of $\frac{1}{3}$, $\frac{2}{6}$, $\frac{3}{9}$, they all can be represented with the co-prime numbers of $(1, 3)$.

Algorithm

We now have the idea and even some important details (co-primes) to implement the algorithm:

- Initiate the maximum number of points `max_count = 1`.
- Iterate over all points `i` from `0` to `N - 2`.
  - For each point `i` find a maximum number of points `max_count_i` on a line passing through the point `i` :
    - Initiate the maximum number of points on a line passing through the point `i` : `count = 1`.
    - Iterate over next points `j` from `i + 1` to `N - 1`.
      - If `j` is a duplicate of `i`, update a number of duplicates for point `i`.
      - If not:
        - Save the line passing through the points `i` and `j`.
        - Update `count` at each step.
    - Return `max_count_i = count + duplicates`.
  - Update the result `max_count = max(max_count, max_count_i)`



Java Python3 Copy

```java
import java.math.BigInteger;

class Solution {
    int[][] points;
    int n;
    HashMap<Pair<Integer, Integer>, Integer> lines = new HashMap<Pair<Integer, Integer>, Integer>();
    int horizontal_lines;

    /**
     * To avoid the precision issue with float/double numbers, using a pair of co-prime numbers to
     * represent the slope.
     */
    private Pair<Integer, Integer> slope_coprime(int x1, int y1, int x2, int y2) {
        int deltaX = x1 - x2, deltaY = y1 - y2;
        if (deltaX == 0)
            return new Pair<>(0, 0);
        else if (deltaY == 0)
            return new Pair<>(Integer.MAX_VALUE, Integer.MAX_VALUE);
        else if (deltaX < 0) {
            deltaX = -deltaX;
            deltaY = -deltaY;
        }
        Integer gcd = BigInteger.valueOf(deltaX).gcd(BigInteger.valueOf(deltaY)).intValue();

        return new Pair<Integer, Integer>(deltaX / gcd, deltaY / gcd);
    }
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N^2)$ since one draws not more than `N - 1` lines passing through the point `0`, not more than `N - 2` lines for the point `1`, and the only one line for the point `N - 2`. That results in `(N - 1) + (N - 2) + .. + 1 = N(N - 1)/2` operations, *i.e.* $\mathcal{O}(N^2)$ time complexity.
- Space complexity : $\mathcal{O}(N)$ to track down not more than `N - 1` lines.

Rate this article:

Comments: 1 Sort By

Type comment here... (Markdown is supported)

Preview Post

bobbydyr ★ 12 July 4, 2020 2:47 AM

The edge cases with deduplicated points make me very upset.

5 Share Reply