Question: https://leetcode.com/problems/max-points-on-a-line/

So the idea is pretty simple, i.e., we consider each point and find out it's slope will all other points which occured after that point in points array.

Now find max number of slope which occured for a single point.

The idea might sound to be simple but the execution isn't! As you all know there is a precision error with floating point numbers, so if we store slope as decimal values and use them as key, then there is high chance we will anomalous result.

So, what to do? We can store the slope as fraction ,i.e.,(x1-x)/(y1-y) but there is a problem with it as well.

we might get 14/21 as slope in a case, and get 4/6 as slope in a case; though both have different values, but the result is same. So, what to do?

We will simplify the fraction to co-prime pairs.

In this case the simplified form would be 2/3, and we acheive that by finding GCD of the numbers and then transforming it to the simplified fraction form.

Code:

public class Solution{

public int maxPoints(int[][] points) {

if (points==null) return 0;

if (points.length<=2) return points.length;

Map<Integer,Map<Integer,Integer>> map = new HashMap<Integer,Map<Integer,Integer>>();

int result=0;

for (int i=0;i<points.length;i++){

map.clear();

int overlap=0,max=0;

for (int j=i+1;j<points.length;j++){

int x=points[j][0]-points[i][0];

int y=points[j][1]-points[i][1];

if (x==0&&y==0){

overlap++;

continue;

}

int gcd=generateGCD(x,y);

if (gcd!=0){

x/=gcd;

y/=gcd;

}

if (map.containsKey(x)){

if (map.get(x).containsKey(y)){

map.get(x).put(y, map.get(x).get(y)+1);

}else{

map.get(x).put(y, 1);

}

}else{

Map<Integer,Integer> m = new HashMap<Integer,Integer>();

m.put(y, 1);

map.put(x, m);

}

max=Math.max(max, map.get(x).get(y));

}

result=Math.max(result, max+overlap+1);

}

return result;

}

private int generateGCD(int a,int b){

if (b==0) return a;

else return generateGCD(b,a%b);

}

}

Github Link :<https://lnkd.in/ecwtJeaz>