Max Points on a Line

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

Solution 1

```
  /*
     *  A line is determined by two factors,say y=ax+b
     *
     *  If two points(x1,y1) (x2,y2) are on the same line(Of course).

     *  Consider the gap between two points.

     *  We have (y2-y1)=a(x2-x1),a=(y2-y1)/(x2-x1) a is a rational, b is canceled
since b is a constant

     *  If a third point (x3,y3) are on the same line. So we must have y3=ax3+b

     *  Thus,(y3-y1)/(x3-x1)=(y2-y1)/(x2-x1)=a

     *  Since a is a rational, there exists y0 and x0, y0/x0=(y3-y1)/(x3-x1)=(y2-
y1)/(x2-x1)=a

     *  So we can use y0&x0 to track a line;
     */

    public class Solution{
        public int maxPoints(Point[] points) {
            if (points==null) return 0;
            if (points.length<=2) return points.length;

            Map<Integer,Map<Integer,Integer>> map = new HashMap<Integer,Map<Integ
er,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].x-points[i].x;
                    int y=points[j].y-points[i].y;
                    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);

        }
    }
```

written by reeclapple original link here

Solution 2

Hint by @stellari

"For each point pi, calculate the slope of each line it forms with all other points with greater indices, i.e. pi+1, pi+2, ..., and use a map to record how many lines have the same slope (If two lines have the same slope and share a common point, then the two lines must be the same one). By doing so, you can easily find how many points are on the same line that ends at pi in O(n). Thus the amortized running time of the whole algorithm is O(n^2)."

In order to avoid using double type(the slope k) as map key, I used pair (int a, int b) as the key where a=pj.x-pi.x, b=pj.y-pi.y, and k=b/a. Using greatest common divider of a and b to divide both a, b ensures that lines with same slope have the same key.

I also handled two special cases: (1) when two points are on a vertical line (2) when two points are the same.

```
class Solution {
public:
    int maxPoints(vector<Point> &points) {

        if(points.size()<2) return points.size();

        int result=0;

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

            map<pair<int, int>, int> lines;
            int localmax=0, overlap=0, vertical=0;

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

                if(points[j].x==points[i].x && points[j].y==points[i].y) {

                    overlap++;
                    continue;
                }
                else if(points[j].x==points[i].x) vertical++;
                else {

                    int a=points[j].x-points[i].x, b=points[j].y-points[i].y;
                    int gcd=GCD(a, b);

                    a/=gcd;
                    b/=gcd;

                    lines[make_pair(a, b)]++;
                    localmax=max(lines[make_pair(a, b)], localmax);
                }

                localmax=max(vertical, localmax);
            }

            result=max(result, localmax+overlap+1);
        }

        return result;
    }

private:
    int GCD(int a, int b) {

        if(b==0) return a;
        else return GCD(b, a%b);
    }
};
```

written by Yoursong original link here

Solution 3

```
int maxPoints(vector<Point> &points) {
    int result = 0;
    for(int i = 0; i < points.size(); i++){
        int samePoint = 1;
        unordered_map<double, int> map;
        for(int j = i + 1; j < points.size(); j++){
            if(points[i].x == points[j].x && points[i].y == points[j].y){
                samePoint++;
            }
            else if(points[i].x == points[j].x){
                map[INT_MAX]++;
            }
            else{
                double slope = double(points[i].y - points[j].y) / double(points[
i].x - points[j].x);
                map[slope]++;
            }
        }
        int localMax = 0;
        for(auto it = map.begin(); it != map.end(); it++){
            localMax = max(localMax, it->second);
        }
        localMax += samePoint;
        result = max(result, localMax);
    }
    return result;
}
```

First, let's talk about mathematics.

How to determine if three points are on the same line?

The answer is to see if slopes of arbitrary two pairs are the same.

Second, let's see what the minimum time complexity can be.

Definitely, O(n^2). It's because you have to calculate all slopes between any two points.

Then let's go back to the solution of this problem.

In order to make this discussion simpler, let's pick a random point A as an example.

Given point A, we need to calculate all slopes between A and other points. There will be three cases:

1. Some other point is the same as point A.
2. Some other point has the same x coordinate as point A, which will result to a positive infinite slope.
3. General case. We can calculate slope.

We can store all slopes in a hash table. And we find which slope shows up mostly. Then add the number of same points to it. Then we know the maximum number of points on the same line for point A.

We can do the same thing to point B, point C...

Finally, just return the maximum result among point A, point B, point C...

written by zxyperfect original link here