Number of Boomerangs

Given *n* points in the plane that are all pairwise distinct, a "boomerang" is a tuple of points `(i, j, k)` such that the distance between `i` and `j` equals the distance between `i` and `k` (**the order of the tuple matters**).

Find the number of boomerangs. You may assume that *n* will be at most **500** and coordinates of points are all in the range **[-10000, 10000]** (inclusive).

Example:

```
Input:
[[0,0],[1,0],[2,0]]

Output:
2

Explanation:
The two boomerangs are [[1,0],[0,0],[2,0]] and [[1,0],[2,0],[0,0]]
```

Solution 1

```
public int numberOfBoomerangs(int[][] points) {
    int res = 0;
    Map<Integer, Integer> map = new HashMap<>();

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

            int d = getDistance(points[i], points[j]);
            map.put(d, map.getOrDefault(d, 0) + 1);
        }

        for(int val : map.values()) {
            res += val * (val-1);
        }
        map.clear();
    }

    return res;
}

private int getDistance(int[] a, int[] b) {
    int dx = a[0] - b[0];
    int dy = a[1] - b[1];

    return dx*dx + dy*dy;
}

Time complexity:  O(n^2)
Space complexity: O(n)
```

written by asurana28 original link here

Solution 2

The idea is simple, for every point, we aggregate points with the same distance and put them in a `Set`.

Update:

I got this solution during the contest, and as troy351 said, Actually we don't need `Map<Integer, Set>`, and `Map<Integer, Integer>` is enough.

```
public class Solution {
    public int numberOfBoomerangs(int[][] points) {
        if(points.length==0 || points[0].length==0) return 0;
        int ret = 0;
        for(int i=0;i<points.length;i++){
            Map<Integer, Set<int[]>> map = new HashMap<>();
            int[] p = points[i];
            for(int j=0;j<points.length;j++){
                if(j==i) continue;
                int[] q = points[j];
                int dis = getDis(p, q);
                if(!map.containsKey(dis)) map.put(dis, new HashSet<int[]>());
                map.get(dis).add(q);
            }
            for(Integer key : map.keySet()){
                int size = map.get(key).size();
                if(size>=2) ret += (size*(size-1));
            }
        }
        return ret;
    }
    public int getDis(int[] p, int[] q){
        int a = p[0]-q[0];
        int b = p[1]-q[1];
        return a*a+b*b;
    }
}
```

written by YuTingLiu original link here

Solution 3

For each point `i`, `map<` distance `d,` count of all points at distance `d` from `i>`.
Given that count, choose `2` (with permutation) from it, to form a boomerang with point `i`.
[use `long` appropriately for `dx`, `dy` and `key`; though not required for the given test cases]

Time Complexity: `O(n^2)`

Updated: Using initial size for the map to avoid table resizing. Thanks @StefanPochmann

```
int numberOfBoomerangs(vector<pair<int, int>>& points) {

    int res = 0;

    // iterate over all the points
    for (int i = 0; i < points.size(); ++i) {

        unordered_map<long, int> group(points.size());

        // iterate over all points other than points[i]
        for (int j = 0; j < points.size(); ++j) {

            if (j == i) continue;

            int dy = points[i].second - points[j].second;
            int dx = points[i].first - points[j].first;

            // compute squared euclidean distance from points[i]
            int key = dy * dy;
            key += dx * dx;

            // accumulate # of such "j"s that are "key" distance from "i"
            ++group[key];
        }

        for (auto& p : group) {
            if (p.second > 1) {
                /*
                 * for all the groups of points,
                 * number of ways to select 2 from n =
                 * nP2 = n!/(n - 2)! = n * (n - 1)
                 */
                res += p.second * (p.second - 1);
            }
        }
    }

    return res;
}
```

written by kdtree original link here

From Leetcoder.