题目：

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]]
```

Subscribe to see which companies asked this question.

1.时间：O（N^2）; 空间：O（N）->结果错误

```
class Solution {
public:
    int numberOfBoomerangs(vector<pair<int, int>>& points) {
        if (points.size() < 3) return 0;
        int result = 0;
        for (int i = 0; i < points.size(); ++i){
            std::unordered_set<int> hashTable;
            for (int k = 0; k < points.size(); ++k){
                if (i == k) continue;
                int distance = calcDist(points[i], points[k]);
                if (hashTable.find(distance) != hashTable.end()){
                    result += 2;
                }
                hashTable.insert(distance);
            }
        }
        return result;
    }
private:
    int calcDist(const std::pair<int, int>& point1, const std::pair<int, int>& point2){
        int dx = point1.first - point2.first;
        int dy = point1.second - point2.second;
        return dx * dx + dy * dy;
    }
};
```

2.时间：O（N^2）; 空间：O（N）

```
class Solution {
public:
    int numberOfBoomerangs(vector<pair<int, int>>& points) {
        if (points.size() < 3) return 0;
        int result = 0;
        for (int i = 0; i < points.size(); ++i){
            std::unordered_map<int, int> hashTable;
            for (int k = 0; k < points.size(); ++k){
                if (i == k) continue;
                int dist = calcDist(points[i], points[k]);
                if (hashTable.find(dist) != hashTable.end()) hashTable[dist]++;
                else hashTable[dist] = 1;
            }
            for (auto it : hashTable){
                int val = it.second;
                result += val * (val - 1);
            }
        }
        return result;
    }
private:
    int calcDist(const std::pair<int, int>& point1, const std::pair<int, int>& point2){
        int dx = point1.first - point2.first;
        int dy = point1.second - point2.second;
        return dx * dx + dy * dy;
    }
};
```