构成的正方形数量题目描述：

输入 N 个互不相同的二维整数坐标，求这 N 个坐标可以构成的正方形数量。(内积为零的两个向量垂直)

输入描述：

第一行输入为 N，N 代表坐标数量，N 为正整数。N <= 100

之后的 K 行输入为坐标 x y 以空格分隔，x, y 为整数，-10<=x, y <= 10

输出描述：

输出可以构成的正方形数量

补充说明：

示例 1

输入：

```
3

1 3

2 4

3 1
```

输出：

```
0
```

说明：

3 个点不足以构成正方形

示例 2

输入：

```
4
0 0
1 2
3 1
2 -1
```

输出：

```
1
```

说明：

此 4 点可构成正方形

```
#include <iostream>

#include <string>

#include <map>

#include <vector>

#include <queue>

#include <cmath>

#include <algorithm>

#include <utility>

#include <random>


#define DEBUG


int g_count = 0;

void travel(std::vector<std::pair<int, int>> &vertexs, std::vector<std::pair<int,

int>> path = std::vector<std::pair<int, int>>())

{
```

```
if (path.empty()) {
    for (auto v : vertexs) {
        std::vector<std::pair<int, int>> next_path;
        next_path.push_back(v);
        travel(vertexs, next_path);
    }
} else if (path.size() == 1) {
    auto u = path.back();
    for (auto v : vertexs) {

        if (u != v) {
            std::vector<std::pair<int, int>> next_path = path;
            next_path.push_back(v);
            travel(vertexs, next_path);
        }
    }
} else if (path.size() <= 4){
    auto v1 = path[path.size() - 1];
    auto v2 = path[path.size() - 2];
    auto vec = std::make_pair(v1.first - v2.first, v1.second - v2.second);
    for (auto v : vertexs) {
        if (v == v1 || v == v2)
```

```
                continue;
            auto vec2 = std::make_pair(v.first - v1.first, v.second - v1.second);
            if (vec2.first * vec.first + vec2.second * vec.second == 0 &&
                vec2.first * vec2.first + vec2.second * vec2.second == vec.first *
vec.first + vec.second * vec.second) {
                std::vector<std::pair<int, int>> next_path = path;
//                  std::cout << "push v: " << v.first << "," << v.second << std::endl;
                next_path.push_back(v);
                travel(vertexs, next_path);
            }
        }
        return;
    } else if (path.size() == 5) {
        if (path[0] == path.back()) {
            g_count++;
        }
        return;
    }

}
int main() {
    int n;
```

```
    std::vector<std::pair<int, int>> vertexs;
    std::cin >> n;

    for (int i = 0; i < n; i++) {
        int x, y;
        std::cin >> x >> y;
        vertexs.push_back(std::make_pair(x, y));
    }

    travel(vertexs);

    std::cout << g_count / 8 << std::endl;


    return 0;
}
//int main() {
//    int color;
//    std::vector<int> list;
//
//    std::cin >> color;
//    std::cin.ignore();
```

```
//
//     std::string str;
//     std::getline(std::cin, str);
////     std::cout << str << std::endl;
//     str.push_back(' ');
//     int head = 0;
//     for (int i = 0; i < str.size(); i++) {
//         if (str[i] == ' ') {
//             std::string s(str, head, i - head);
//             if (!s.empty()) {
//                 list.push_back(std::stoi(s));
//             }
//             head = i + 1;
//         }
//     }
//
////     for (auto n : list) {
////         std::cout << n << " ";
////     }
////     std::cout << std::endl;
//
//     int max_count = 0;
```

```
//     int cur_count = 0;
//     int nega_count = 0;
//     int pos = -1;
//     for (int i = 0; i < list.size(); i++) {
//         if (list[i] == color) {
//             nega_count = 0;
//             cur_count++;
//         } else if (list[i] == -color) {
//             cur_count = 0;
//             nega_count++;
//         } else {
//             cur_count++;
//             for (int j = i + 1; j < list.size(); j++) {
//                 if (list[j] == color) {
//                     cur_count++;
//                 } else {
//                     break;
//                 }
//             }
//             if (cur_count > max_count && cur_count <= 5) {
//                 pos = i;
//                 max_count = cur_count;
```

```
//             } else if (cur_count == max_count) {
//                             if (std::abs(pos - (int)(list.size()/2)) > std::abs(i -
(int)(list.size()/2))) {
//                         pos = i;
//                     } else if (std::abs(pos - (int)(list.size()/2)) == std::abs(i -
(int)(list.size()/2))) {
//                         pos = pos < i ? pos : i;
//                     }
//             }
//             cur_count = 0;
//             nega_count = 0;
//         }
//     }
//
//     std::cout << pos << std::endl;
//
//
//     return 0;
//}
```