codility

g center

lity training tasks

**Congratulations**

You have completed a Codility training test.

Tweet this!
I scored 100% in #python on @Codility!
https://codility.com/demo/take-sample-test/diamonds_cour

Sign up for our newsletter!

Like us on Facebook!

# *Training ticket*

## Session

**ID:** training2Q232V-FZN
**Time limit:** 120 min.

## Status: closed

**Created on:** 2017-07-02 04:30 UTC
**Started on:** 2017-07-02 04:30 UTC
**Finished on:** 2017-07-02 05:40 UTC

| Tasks in test | Correctness | Performance | Task score |
|---|---|---|---|
| 1 DiamondsCount<br>Submitted in: Python | 100% | 100% | 100% |

Test score

**100%**

100 out of 100 points

HARD

## 1. DiamondsCount

score: 100 of 100

Given points on a plane, count the number of sets of four points that form regular diamonds.

### Task description

A *diamond* is a quadrilateral whose four sides all have the same length and whose diagonals are parallel to the coordinate axes.

You are given N distinct points on a plane. Count the number of different diamonds that can be constructed using these points as vertices (two diamonds are different if their sets of vertices are different). Do not count diamonds whose area is empty.

Write a function:

```
def solution(X, Y)
```

that, given two zero-indexed arrays X and Y, each containing N integers, representing N points (where X[K], Y[K] are the coordinates of the K-th point), returns the number of diamonds on the plane.

For example, for N = 7 points whose coordinates are specified in arrays X = [1, 1, 2, 2, 2, 3, 3] and Y = [3, 4, 1, 3, 5, 3, 4], the function should return 2, since we can find two diamonds as shown in the picture below:

### Solution

**Programming language used:** Python

**Total time used: 71 minutes**

**Effective time used: 71 minutes**

**Notes:** *not defined yet*

**Task timeline**

04:30:43

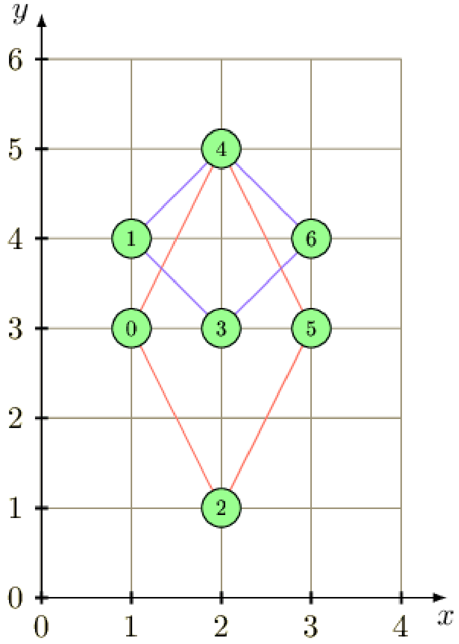
05:40:47

Code: 05:40:47 UTC, py, final, score: **100**

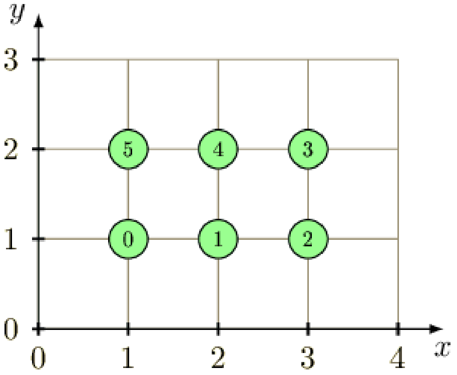
show code in pop-up

```
# you can write to stdout for debugging purposes, e.g.
# print "this is a debug message"

def solution(X, Y):
    N = len(X)
    grid = [[0 for j in xrange(N)] for i in xrange(N)]
    rows = {}
    cols = {}
```



Given arrays: X = [1, 2, 3, 3, 2, 1], Y = [1, 1, 1, 2, 2, 2], the function should return 0, since there are no diamonds on the plane:



Assume that:

- N is an integer within the range [4..1,500];
- each element of arrays X, Y is an integer within the range [0..N−1];
- given N points are pairwise distinct.

Complexity:

- expected worst-case time complexity is O(N$^2$);
- expected worst-case space complexity is O(N$^2$), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

```
    for i in xrange(N):
        grid[X[i]][Y[i]] = 1
        if X[i] not in rows:
            rows[X[i]] = set([Y[i]])
        else:
            rows[X[i]].add(Y[i])
        if Y[i] not in cols:
            cols[Y[i]] = set([X[i]])
        else:
            cols[Y[i]].add(X[i])
    counter = 0
    for i in xrange(N):
        x = X[i]
        y = Y[i]
        y_col = cols[y]
        for row_point in y_col:
            if row_point > x and (row_point - x) % 2 ==
                middle = x + (row_point - x)//2
                if middle in rows:
                    mid_row = rows[middle]
                    for mid_col in mid_row:
                        if mid_col > y:
                            other_side = y - (mid_col -
                            if other_side >= 0:
                                if grid[middle][other_
                                    counter += 1
    return counter

```

### Analysis summary

The solution obtained perfect score.

### Analysis

Detected time complexity:
O(N**2)

collapse all Example tests

▼ example1 ✔ OK
first example test

1. 0.070 s OK

▼ example2 ✔ OK
second example test

1. 0.065 s OK

expand all Correctness tests

▶ tiny ✔ OK
tiny test with N = 4 points

▶ small_full_square ✔ OK
N = 25 points forming full square

▶ small_random ✔ OK
random small tests, 5 <= N <= 60 points

▶ small_one_line ✔ OK
small tests with all points in one line

expand all Performance tests

▼ max_dense ✔ OK
dense max tests

1. 0.207 s OK

2. 0.214 s OK

3. 0.185 s OK

▶ max_random ✔ OK
random max tests

▼

max_one_line ✔ OK

max tests with many points in one line

1. 0.427 s **OK**

2. 0.180 s **OK**

▶ max_cross_square ✔ OK

points arranged in a cross or square

▶ max_three_lines ✔ OK

max tests with points in three parallel lines