Clocks

It's time to show how the Codility Chalenge codenamed `Li` (Lithium) can be solved. You can still give it a try, but no certificate will be granted.

The story is about round clocks. The clocks have round faces and a number of identical hands. The goal is to rotate them so that the number of pairs of clocks that look identical is maximum.

Brute-force solution

For every pair of clocks, we will check whether they can look identical or not. Let's sort the hands of each clock. After that, we can easily count the distances between consecutive hands. Two clocks look identical if the first hands are aligned and the consecutive distances between hands are the same. When we rotate a clock, some other hand can become *the first one*. So to check whether two clocks can be rotated to look identically, we have to check every cyclic rotation of the distances between consecutive hands.

Here is a program comparing all the pairs of clocks and checking whether they can be rotated so that they look identically. Assume that matrix A contains distances between consecutive hands.

1: Brute-force solution.

```
def clocks(A):
    N = len(A)
    M = len(A[0])
    result = 0
    for i in xrange(N):
        for k in xrange(i + 1, N):
            for l in xrange(M):
                ok = True
                for j in xrange(M):
                    if (A[i][j] != A[k][(j + l) % M]):
                        ok = False
                        break;
                if ok:
                    result += 1
                    break;
    return result
```

However, the above solution is inefficient. We have $O(N^2)$ pairs of clocks, and every pair is compared in $O(M^2)$ time, so the whole algorithm may require $O(N^2 \cdot M^2)$ time.

Slow solution

To find a better solution, we should be able to compare two clocks faster than in $O(M^2)$ time. There is a suitable algorithm called lexicographically minimal string rotation. For each clock, we find its canonical rotation. If two clocks can be rotated so that look identically, their canonical rotations should be identical. For this purpose we can choose such a rotation of distances between consecutive hands that is minimal in the lexicographical order. Finding the minimal rotation of a clock requires $O(M)$ time, so the time complexity of the whole algorithm is $O(N^2 \cdot M + N \cdot M \log M)$.

2: Slow solution.

```
def clocks(A):
    N = len(A)
    M = len(A[0])
    for i in xrange(N):
        minimal_lexicographically_rotation(A[i])
    result = 0
    for i in xrange(N):
        for k in xrange(i + 1, N):
            ok = True
            for j in xrange(M):
                if (A[i][j] != A[k][j]):
                    ok = False
                    break;
            if ok:
                result += 1
    return result
```

Optimal solution

We can find an even faster solution to this task by simply sorting the clocks in the order of their lexicographically minimal rotations. That will cause identical clocks to be adjacent to each other in the array. Assume that matrix A contains distances between consecutive hands.

3: Golden solution.

```
def clocks(A):
    N = len(A)
    for i in xrange(N):
        minimal_lexicographically_rotation(A[i])
    A.sort()
    result = 0
    pairs = 0
    for i in xrange(1, N):
        if (equal(A[i], A[i - 1])):
            pairs += 1
            result += pairs
        else:
            pairs = 0
    return result
```

With this approach, the time complexity is $O(N \cdot M \cdot (\log M + \log N))$. This solution should achieve the maximum number of points.

Alternative solution

We can also solve this task in alternative way. Instead of looking for lexicographically minimal rotations, we can find the rotation in which we get the maximal (or minimal) hash. If the maximal hash for two clocks is the same, we claim that the two clocks look identical.

Finally, just sort the clocks by their maximal hashes and identical clocks will also be next to each other. The time complexity is $O(N \cdot M \cdot \log M + N \cdot \log N)$.