Daily Coding Problem #68

Problem

This problem was asked by Google.

On our special chessboard, two bishops attack each other if they share the same diagonal. This includes bishops that have another bishop located between them, i.e. bishops can attack through pieces.

You are given N bishops, represented as (row, column) tuples on a M by M chessboard. Write a function to count the number of pairs of bishops that attack each other. The ordering of the pair doesn't matter: (1, 2) is considered the same as (2, 1).

For example, given M = 5 and the list of bishops:

- (0, 0)
- (1, 2)

- (2, 2)
- (4, 0)

The board would look like this:

```
[b 0 0 0 0]
[0 0 b 0 0]
[0 0 b 0 0]
[0 0 0 0 0]
[b 0 0 0 0]
```

You should return 2, since bishops 1 and 3 attack each other, as well as bishops 3 and 4.

Solution

One approach would be to iterate through each bishop and find all the other attacking bishops, incrementing the count when we find a pair.

We can define a helper function `is_attacking` that returns whether or not two bishops are attacking each other:

```
def is_attacking(bishop0, bishop1):
    r0, c0 = bishop0
    r1, c1 = bishop1
    return abs(r1 - r0) == abs(c1 - c0)

def pairs(bishops, m):
    count = 0
    for i, bishop0 in enumerate(bishops):
        for j, bishop1 in enumerate(bishops[i + 1:]):
            count += is_attacking(bishop0, bishop1)
```

```
    return count
```

This would take O(N^2). Can we make this any faster?

If we know how many bishops are in each diagonal, then we can know how many pairs are attacking: for each diagonal, it's the number of bishops choose 2, since each bishop makes a pair with every other bishop.

So, if we go through each bishop and bucket them into each separate diagonal, we can just run (b choose 2) on the number of bishops on each diagonal and sum them up. Recall that (n choose 2) is equivalent to `n * (n - 1) / 2`.

Each *bucket* is represented by a tuple `top_left_row, top_left_column, direction`. (Or right row if it's the other way.) Then we can quickly figure out which bucket a bishop belongs to by moving up each diagonal until we hit a border.

```
from collections import defaultdict

TOP_LEFT_TO_BOTTOM_RIGHT = 0
TOP_RIGHT_TO_BOTTOM_LEFT = 1

def combos(num):
    return num * (num - 1) / 2

def pairs(bishops, m):
    counts = defaultdict(int)
    for r, c in bishops:
        top_lr, top_lc = (r - min(r, c), c - min(r, c))
        top_rr, top_rc = (r - min(r, m - c), c + min(r, m - c))

        counts[top_lr, top_lc, TOP_LEFT_TO_BOTTOM_RIGHT] += 1
        counts[top_rr, top_rc, TOP_RIGHT_TO_BOTTOM_LEFT] += 1
    return sum(combos(c) for c in counts.values())
```

This runs in O(N) time and space.