Daily Coding Problem #269

Problem

This problem was asked by Microsoft.

You are given an string representing the initial conditions of some dominoes. Each element can take one of three values:

- `L`, meaning the domino has just been pushed to the left,
- `R`, meaning the domino has just been pushed to the right, or
- `.`, meaning the domino is standing still.

Determine the orientation of each tile when the dominoes stop falling. Note that if a domino receives a force from the left and right side simultaneously, it will remain upright.

For example, given the string `.L.R....L`, you should return `LL.RRRLLL`.

Given the string `..R...L.L`, you should return `..RR.LLLL`.

Solution

Each domino can only be acted on by at most two other dominoes: one applying a force from the left, and the other applying a force from the right. For example, in the second string above, the last `L` has no impact on any tile before the first `L`. As a result, we can

consider each section between the initally pushed tiles to be independent, and apply logic to each of these in turn.

To find the boundaries of each section, we can iterate through our domino string, adding the low and high index of each pair of nearby initially pushed tiles. We must take special care to include boundaries at the beginning and end, to handle edges cases where a chain of dominoes gets pushed all the way to the end.

```
def get_boundaries(dominoes):
    boundaries = []

    low = high = 0
    for high, tile in enumerate(dominoes[1:], 1):
        if tile != '.':
            boundaries.append((low, high))
            low = high

    if boundaries[-1][-1] != len(dominoes) - 1:
        boundaries.append((low, high))

    return boundaries
```

This will take `O(N)` time and space, since we must iterate over all dominoes and create a new list that may include every tile.

Once we have these boundaries, we can update the values of the tiles within each section. There are four cases we need to check:

- If the section begins at the left edge and goes to an `L`, the dominoes will all fall left.
- If the section begins with an `R` and goes to the right edge, the dominoes will all fall right.
- If the section begins and ends with the same value, all the dominoes in between should fall in that direction.
- If the section begins with an `R` and ends with an `L`, the tiles in between should fall towards the center.

In code, this would look like the following:

```
def push(dominoes):
    boundaries = get_boundaries(dominoes)
    dominoes = list(dominoes)

    for low, high in boundaries:
        if dominoes[low] == '.' and dominoes[high] == 'L':
            dominoes[low : high] = ['L'] * (high - low)

        elif dominoes[low] == 'R' and dominoes[high] == '.':
            dominoes[low : high + 1] = ['R'] * (high + 1 - low)

        elif dominoes[low] == dominoes[high]:
            dominoes[low : high] = dominoes[low] * (high - low)

        elif dominoes[low] == 'R' and dominoes[high] == 'L':
            while low + 1 < high - 1:
                dominoes[low + 1] = dominoes[low]
                dominoes[high - 1] = dominoes[high]
                low += 1; high -= 1

    return ''.join(dominoes)
```

For each boundary, we are updating at most a number of tiles equal to the size of the section. As a result, the number of operations will be on the order of the number of total dominoes. And similar to the function above, we must allocate additional space, in this case for the updated domino array. As a result, the entire algorithm runs in `O(N)` time and space.

Press