Daily Coding Problem #290

Problem

This problem was asked by Facebook.

On a mysterious island there are creatures known as Quxes which come in three colors: red, green, and blue. One power of the Qux is that if two of them are standing next to each other, they can transform into a single creature of the third color.

Given `N` Quxes standing in a line, determine the smallest number of them remaining after any possible sequence of such transformations.

For example, given the input `['R', 'G', 'B', 'G', 'B']`, it is possible to end up with a single Qux through the following steps:

```
        Arrangement       |   Change
----------------------------------------
['R', 'G', 'B', 'G', 'B'] | (R, G) -> B
['B', 'B', 'G', 'B']      | (B, G) -> R
['B', 'R', 'B']           | (R, B) -> G
['B', 'G']                | (B, G) -> R
['R']                     |
```

Solution

Solution

One way to solve this problem would be to try every possible sequence of transformations, and find the one that results in the smallest result.

We can make this approach a little more methodical using recursion. For a given starting array, any transformation will turn the array into a new one with one fewer element, to which we can then re-apply our function. At each step of our process we can apply our function to all possible transformation and return the minimal outcome.

The base case for this recursion is when all the Quxes in the line are the same color. In this case no moves are possible, so we must return the length of the line.

```
COLORS = {'R', 'G', 'B'}

def transform(a, b):
    return list(COLORS - {a, b})

def num_remaining(quxes):
    if len(set(quxes)) == 1:
        return len(quxes)

    results = []
    for i, pair in enumerate(zip(quxes, quxes[1:])):
        if pair[0] != pair[1]:
            results.append(num_remaining(quxes[:i] + transform(*pair) + quxes[i + 2:]))

    return min(results)
```

For any array, there may be `N - 1` possible transformations to apply, where `N` is the length of the array. As a result, we may need to call our function `(N - 1) * (N - 2) * ... * 1` times, as the number of possible sequences proliferates. The time of complexity of this algorithm is therefore `O(N!)`.

Fortunately, we can use math to find a more elegant solution.

First, recall that any integer must be either even or odd, a quality known as its parity. Now suppose we have three integers `(a, b, c)`, representing the number of Quxes of each color. For any given configuration there will be four cases:

- (a) These numbers are all even

- (a) These numbers are all even
- (b) These numbers are all odd
- (c) Two are even, and one is odd
- (d) Two are odd, and one is even

The cases fall into two groups: either the parity of all the integers will be the same (a and b), or the parity will break down into two numbers on one side, and one on the other (c and d).

Now note that for any transformation, we reduce the count of two colors by one, and increase the value of the other by one. For example, in the change `(R, G) -> B`, we decrement the number of red and green Quxes and increment the number of blue ones. Crucially, this does not change which of the two above groups the line falls into. In other words, a Qux line will always be alternating between cases a and b, or between c and d, but never both.

What, then, is the best possible outcome for each group? For the first group, this will be either `(2, 0, 0)`, `(0, 2, 0)`, or `(0, 0, 2)`. That is, once we reach a point where we have two Quxes of one color and none of any other color, no new transformations are possible. And since it is impossible to remove all Quxes from the line, these are the equal-parity formations with the lowest sum.

Analogously, the best outcome for a split-parity group will be `(1, 0, 0)`, `(0, 1, 0)`, or `(0, 0, 1)`.

With some care we can show that as long as our initial line does not consist of all Quxes of the same color, we can continually apply transformations to get to one of these base cases. As a result, our solution will simply be the following:

- If all Quxes begin as the same color, return the length of the line.
- If the parities of each color are equal, return 2.
- If the parities of each color are split, return 1.

```
def num_remaining(quxes):
    if len(set(quxes)) == 1:
        return len(quxes)

    counts = {'R': 0, 'B': 0, 'G': 0}
    for qux in quxes:
        counts[quxes] += 1
```

```
    if counts['R'] % 2 == counts['B'] % 2 == counts['G'] % 2:
        return 2

    else:
        return 1
```

The time and space complexity of this solution is `O(N)`, since all we need to do is count up the Qux colors and calculate each count `mod 2`.
