Daily Coding Problem #216

Problem

This problem was asked by Facebook.

Given a number in Roman numeral format, convert it to decimal.

The values of Roman numerals are as follows:

```
{
    'M': 1000,
    'D': 500,
    'C': 100,
    'L': 50,
    'X': 10,
    'V': 5,
    'I': 1
}
```

In addition, note that the Roman numeral system uses subtractive notation for numbers such as `IV` and `XL`.

For the input `XIV`, for instance, you should return `14`.

Solution

Let's look at an example Roman numeral, say `DCCCXLI`.

To compute the value of this number in our heads, we would proceed left to right, increasing our total along the way. Specifically, we starting from `0`, we would:

- Add `500` (for `D`)
- Add `100` (for `C`)

- Add `100` (for `C`)
- Add `100` (for `C`)
- Add `40` (for `XL`)
- Add `1` (for `I`)
- After running out of letters, return `841`.

In other words, all the letters except for the pair `XL` were directly evaluated and added to our running sum. In the latter case, mapping the two-letter pair took priority over evaluating the letters separately. This points the way to a solution: in addition to the dictionary given in the problem description, we can create another enumerating all two-letter exceptions.

As we proceed along the string, we check if the first two letters exist in our pair dictionary. If so, we add the appropriate value to our total. Otherwise, we add the value of just the first letter. In either case, we repeat this process on the remainder of the string, until we run out of letters. It may be helpful to note that all two-letter pairs correspond to values of the form `4 * 10`^{N} or `9 * 10`^{N}.

```
PAIRS = {
    'CM': 900,
    'CD': 400,
    'XC': 90,
    'XL': 40,
    'IX': 9,
    'IV': 4
}

SINGLES = {
    'M': 1000,
    'D': 500,
    'C': 100,
    'L': 50,
    'X': 10,
    'V': 5,
    'I': 1
}

def decimate(s, total=0):
    if not s:
        return total
    if s[:2] in PAIRS:
        total += PAIRS[s[:2]]
        return decimate(s[2:], total)
    else:
        total += SINGLES[s[:1]]
        return decimate(s[1:], total)
```

This works, but what if we weren't allowed to use an extra dictionary?

Let's take a closer look at what's happening when we see a pair such as `XL`. Note that for each of these pairs, the first letter's value is less than that of the second letter. Additionally, instead of adding `40`, we can equivalently subtract `10` (`X`), and then add `50` (`L`). Putting these ideas together, if we come across a case where `s[i] < s[i + 1]`, we can subtract the first letter and proceed normally on the rest of the string. This obviates the need for the second dictionary and gives us a simple iterative solution.

```
def decimate(s):
    decimal_map = {'M': 1000, 'D': 500, 'C': 100, 'L': 50, 'X': 10, 'V': 5, 'I': 1}

    total = 0
    for i in range(len(s) - 1):
        if decimal_map[s[i]] >= decimal_map[s[i + 1]]:
            total += decimal_map[s[i]]
        else:
            total -= decimal_map[s[i]]
    total += decimal_map[s[-1]]

    return total
```

Both of these algorithms are linear with respect to the number of letters in the input.