800. Similar RGB Color

March 17, 2018 | 7.3K views

Average Rating: 2 (35 votes)

In the following, every capital letter represents some hexadecimal digit from `0` to `f`.

The red-green-blue color `"#AABBCC"` can be written as `"#ABC"` in shorthand. For example, `"#15c"` is shorthand for the color `"#1155cc"`.

Now, say the similarity between two colors `"#ABCDEF"` and `"#UVWXYZ"` is `-(AB - UV)^2 - (CD - WX)^2 - (EF - YZ)^2`.

Given the color `"#ABCDEF"`, return a 7 character color that is most similar to `#ABCDEF`, and has a shorthand (that is, it can be represented as some `"#XYZ"`

```
Example 1:
Input: color = "#09f166"
Output: "#11ee66"
Explanation:
The similarity is -(0x09 - 0x11)^2 -(0xf1 - 0xee)^2 - (0x66 - 0x66)^2 = -64 -9 -0 = -7
This is the highest among any shorthand color.
```

Note:

- `color` is a string of length `7`.
- `color` is a valid RGB color: for `i > 0`, `color[i]` is a hexadecimal digit from `0` to `f`
- Any answer which has the same (highest) similarity as the best answer will be accepted.
- All inputs and outputs should use lowercase letters, and the output is 7 characters.

Approach #1: Brute Force [Accepted]

Intuition

For each possible shorthand-RGB color from `"#000"` to `"#fff"`, let's find it's similarity to the given color. We'll take the best one.

Algorithm

This problem is straightforward, but there are a few tricky implementation details.

To iterate over each shorthand color, we'll use an integer based approach, (though other ones exist.) Each digit in the shorthand `"#RGB"` could be from 0 to 15. This leads to a color of 17 * R * (1 << 16) + 17 * G * (1 << 8) + 17 * B. The reason for the 17 is because a hexadecimal value of `0x22` is equal to `2 * 16 + 2 * 1` which is `2 * (17)`. The other values for red and green work similarly, just shifted up by 8 or 16 bits.

To determine the similarity between two colors represented as integers, we'll sum the similarity of their colored components separately. For a color like `hex1`, it has 3 colored components `r1 = (hex1 >> 16) % 256, g1 = (hex1 >> 8) % 256, b1 = (hex1 >> 0) % 256`. Then, the first addend in the similarity is `-(r1 - r2) * (r1 - r2)`, etc.

To convert an integer back to a hex string, we'll use `String.format`. The `06` refers to a zero padded string of length 6, while `x` refers to lowercase hexadecimal.

Finally, it should be noted that the answer is always unique. Indeed, for two numbers that differ by 17, every integer is closer to one than the other. For example, with `0` and `17`, `8` is closer to `0` and `9` is closer to `17` - there is no number that is tied in closeness.

Java | Python — Copy

```python
class Solution(object):
    def similarRGB(self, color):
        def similarity(hex1, hex2):
            r1, g1, b1 = hex1 >> 16, (hex1 >> 8) % 256, hex1 % 256
            r2, g2, b2 = hex2 >> 16, (hex2 >> 8) % 256, hex2 % 256
            return -(r1 - r2)**2 - (g1 - g2)**2 - (b1 - b2)**2

        hex1 = int(color[1:], 16)
        ans = 0
        for r in xrange(16):
            for g in xrange(16):
                for b in xrange(16):
                    hex2 = 17 * r * (1 << 16) + 17 * g * (1 << 8) + 17 * b
                    if similarity(hex1, hex2) > similarity(hex1, ans):
                        ans = hex2

        return '#{:06x}'.format(ans)
```

Complexity Analysis

- Time and Space Complexity: $O(1)$.

Approach #2: Rounding By Component [Accepted]

Intuition and Algorithm

Because color similarity is a sum of the similarity of individual color components, we can treat each colored component separately and combine the answer.

As in the previous approach, we want the colored component to be the closest to a multiple of 17. We can just round it to the closest such value.

Java | Python — Copy

```python
class Solution(object):
    def similarRGB(self, color):
        def f(comp):
            q, r = divmod(int(comp, 16), 17)
            if r > 8: q += 1
            return '{:02x}'.format(17 * q)

        return '#' + f(color[1:3]) + f(color[3:5]) + f(color[5:])
```

Complexity Analysis

- Time and Space Complexity: $O(1)$.

Analysis written by: @awice.

Rate this article:

Previous Next

Comments: 8

Sort By

Type comment here... (Markdown is supported)

Preview Post

JuniorOgun ★ 33 November 18, 2018 5:16 AM

Is this really a easy problem seems closer to medium?

25 Share Reply

DennyZhang ★ 220 March 19, 2018 7:09 AM

For Brute Force, do we really need 3 loop?

To get a * a+b * b+c * c smallest, we can make sure a, b, c itself is the smallest.

So we can lower the complexity from O(16 * 16 * 16) to O(3 * 16).

Read More

7 Share Reply

SHOW 1 REPLY

shubham75 ★ 6 March 19, 2018 9:00 AM

What is the logic behind this step in second approach:

(q % 17 > 8 ? 1 : 0)

3 Share Reply

SHOW 1 REPLY

966540 ★ 7 January 2, 2019 11:55 PM

in approach 2: why is 17 not 16?? 11(hex) = 17(decl) 22 = 32 + 2 = 34

2 Share Reply

zerustech ★ 200 December 20, 2018 1:43 PM

An explanation of approach 2

Assume `a = 17 * x + y, b = 17 * z`, and `a` is the color component, `b` is the target value we are

Read More

2 Share Reply

SHOW 1 REPLY

ye15 ★ 326 November 16, 2019 4:35 AM

Not as elegant as the solution, but one could also pick the closest from (at most) three candidates for the given color component like below

```
class Solution:
    def similarRGB(self, color: str) -> str:
```

Read More

1 Share Reply

leetcode_deleted_user ★ 246 April 6, 2018 9:44 AM

q = q / 17 + (q % 17 > 8 ? 1 : 0); want to know how this method work to find out which color component will be used? Thanks

0 Share Reply

SHOW 2 REPLIES

aciash2009 ★ 11 August 6, 2019 6:31 AM

nice solution!

0 Share Reply