681. Next Closest Time

Sept. 27, 2017 | 48.5K views

Average Rating: 2.86 (44 votes)

Given a time represented in the format "HH:MM", form the next closest time by reusing the current digits. There is no limit on how many times a digit can be reused.

You may assume the given input string is always valid. For example, "01:34", "12:09" are all valid. "1:34", "12:9" are all invalid.

Example 1:

```
Input: "19:34"
Output: "19:39"
Explanation: The next closest time choosing from digits 1, 9, 3, 4, is 19:39, which oc
```

Example 2:

```
Input: "23:59"
Output: "22:22"
Explanation: The next closest time choosing from digits 2, 3, 5, 9, is 22:22. It may h
```

Approach #1: Simulation [Accepted]

Intuition and Algorithm

Simulate the clock going forward by one minute. Each time it moves forward, if all the digits are allowed, then return the current time.

The natural way to represent the time is as an integer `t` in the range `0 <= t < 24 * 60`. Then the hours are `t / 60`, the minutes are `t % 60`, and each digit of the hours and minutes can be found by `hours / 10`, `hours % 10` etc.

Java Python Copy

```
class Solution(object):
    def nextClosestTime(self, time):
        cur = 60 * int(time[:2]) + int(time[3:])
        allowed = {int(x) for x in time if x != ':'}
        while True:
            cur = (cur + 1) % (24 * 60)
            if all(digit in allowed
                   for block in divmod(cur, 60)
                   for digit in divmod(block, 10)):
                return "{:02d}:{:02d}".format(*divmod(cur, 60))
```

Complexity Analysis

- Time Complexity: $O(1)$. We try up to $24 * 60$ possible times until we find the correct time.
- Space Complexity: $O(1)$.

Approach #2: Build From Allowed Digits [Accepted]

Intuition and Algorithm

We have up to 4 different allowed digits, which naively gives us `4 * 4 * 4 * 4` possible times. For each possible time, let's check that it can be displayed on a clock: ie., `hours < 24 and mins < 60`. The best possible `time != start` is the one with the smallest `cand_elapsed = (time - start) % (24 * 60)`, as this represents the time that has elapsed since `start`, and where the modulo operation is taken to be always non-negative.

For example, if we have `start = 720` (ie. noon), then times like `12:05 = 725` means that `(725 - 720) % (24 * 60) = 5` seconds have elapsed; while times like `00:10 = 10` means that `(10 - 720) % (24 * 60) = -710 % (24 * 60) = 730` seconds have elapsed.

Also, we should make sure to handle `cand_elapsed` carefully. When our current candidate time `cur` is equal to the given starting time, then `cand_elapsed` will be `0` and we should handle this case appropriately.

Java Python Copy

```
class Solution(object):
    def nextClosestTime(self, time):
        ans = start = 60 * int(time[:2]) + int(time[3:])
        elapsed = 24 * 60
        allowed = {int(x) for x in time if x != ':'}
        for h1, h2, m1, m2 in itertools.product(allowed, repeat = 4):
            hours, mins = 10 * h1 + h2, 10 * m1 + m2
            if hours < 24 and mins < 60:
                cur = hours * 60 + mins
                cand_elapsed = (cur - start) % (24 * 60)
                if 0 < cand_elapsed < elapsed:
                    ans = cur
                    elapsed = cand_elapsed

        return "{:02d}:{:02d}".format(*divmod(ans, 60))
```

Complexity Analysis

- Time Complexity: $O(1)$. We all 4^4 possible times and take the best one.
- Space Complexity: $O(1)$.

Analysis written by: @awice

Rate this article:

Previous Next

Comments: 30

Sort By

Type comment here... (Markdown is supported)

Preview Post

yhzs8 ★ 72 January 17, 2018 1:35 PM

The following line in the 2nd solution seems to be overcomplicated and not necessary to use floored modulus or normal modulus (especially the floored modulus is not very common and hard to understand/explain/use):

Read More

23 Share Reply

logical_paradox ★ 238 October 15, 2018 3:40 AM Report

I think you're talking about minutes in the entire discussion. It is misleading if you're talk seconds and then calculate using minutes.

12 Share Reply

Cloudson ★ 90 October 7, 2018 9:44 PM

Can someone explain the following expression to me? Is the "search :{ break search}" a new characteristic of JAVA? Thank you!

search : {
for (int d: digits) if (!allowed.contains(d)) break search;

Read More

6 Share Reply

SHOW 2 REPLIES

yentup ★ 104 August 17, 2018 12:16 AM

If we sort the digits, then we can break early (since all subsequent times will be invalid).

```
allowed = sorted({int(x) for x in time if x != ':'})
```

Read More

4 Share Reply

colazil ★ 3 November 19, 2018 4:46 PM Report

for solution2 ,we can run faster by build allowed hour and minute separately, this can reduce time to 4 x 4 x 2

```
def nextClosestTime(self, time):
```

Read More

3 Share Reply

jaege ★ 28 October 13, 2018 11:10 PM

A even simpler approach (I think):

```
string nextClosestTime(string time) {
    set<char> digits(time.begin(), time.end());
    digits.erase(':');
    auto it = digits.end();
    vector<function<bool()>> cmp{
        [&it]{ return *it < '3'; },                 // H 0~1 2
        [&it, &time]{ return time[0] < '2' || *it < '4'; },  // H 0~9 0~3
        []{ return true; },                         // :
        [&it]{ return *it < '6'; },                 // M 0~5
        []{ return true; },                         // M 0~9
    };
    for (int i = 4; i >= 0; --i) {
        if (i == 2) continue;
        it = digits.find(time[i]);
        if (++it != digits.end() && cmp[i]()) {
            time[i] = *it;
            return time;
        }
        time[i] = *digits.begin();
    }
    return time;
}
```

3 Share Reply

mahekjasani ★ 29 January 9, 2018 11:40 AM

Why don't we break the loop as soon as we find our answer?

2 Share Reply

SHOW 1 REPLY

lch04 ★ 8 January 1, 2018 5:13 AM

Typos in the 2nd solution: (725 - 720) % (24 * 60) = 5 minutes instead of (725 - 720) % (24 * 60) = 5 seconds

2 Share Reply

mehranangelo ★ 108 December 17, 2018 8:53 AM

Interview Friendly Solution using TreeSet

```
public String nextClosestTime(String time) {
    int n=time.length();
```

Read More

3 Share Reply

volodymyrost ★ 1 March 29, 2019 12:41 PM Report

Here is my much faster approach. C++ only.

The algorithm mimics how human solves this.

The main idea is to explore few facts:

Read More

1 Share Reply

1 2 3