681. Next Closest Time (/problems/next-closest-time/)

Sept. 27, 2017 | 4.3K views

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 occurs 5 minu
```

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 be assumed t
```

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

```java
class Solution {
    public String nextClosestTime(String time) {
        int cur = 60 * Integer.parseInt(time.substring(0, 2));
        cur += Integer.parseInt(time.substring(3));
        Set<Integer> allowed = new HashSet();
        for (char c: time.toCharArray()) if (c != ':') {
            allowed.add(c - '0');
        }

        while (true) {
            cur = (cur + 1) % (24 * 60);
            int[] digits = new int[]{cur / 60 / 10, cur / 60 % 10, cur % 60 / 10, cur % 60 % 10};
            search : {
                for (int d: digits) if (!allowed.contains(d)) break search;
                return String.format("%02d:%02d", cur / 60, 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

```java
class Solution {
    public String nextClosestTime(String time) {
        int start = 60 * Integer.parseInt(time.substring(0, 2));
        start += Integer.parseInt(time.substring(3));
        int ans = start;
        int elapsed = 24 * 60;
        Set<Integer> allowed = new HashSet();
        for (char c: time.toCharArray()) if (c != ':') {
            allowed.add(c - '0');
        }

        for (int h1: allowed) for (int h2: allowed) if (h1 * 10 + h2 < 24) {
            for (int m1: allowed) for (int m2: allowed) if (m1 * 10 + m2 < 60) {
                int cur = 60 * (h1 * 10 + h2) + (m1 * 10 + m2);
                int candElapsed = Math.floorMod(cur - start, 24 * 60);
                if (0 < candElapsed && candElapsed < elapsed) {
                    ans = cur;
                    elapsed = candElapsed;
                }
            }
        }

        return String.format("%02d:%02d", ans / 60, 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 (https://leetcode.com/awice)

Ark-kun commented last month

There is a faster way to do this. The time is less than 4*4 instead of 4^4.

(https://discuss.leetcode.com/user/ark-kun)

Midgar77 commented last month

@awice (https://discuss.leetcode.com/uid/71269) I think the following sentence accidentally has a repeated part:

"while times like 00:10 = 10 means that (10 - 720) % (24 * 60) = -710 % (24 * 60) = 730 seconds have elapsed."

I think you want to take out the first "% (24 * 60)" so it reads:
"while times like 00:10 = 10 means that (10 - 720) = -710 % (24 * 60) = 730 seconds have elapsed."

Just trying to help, great answer! Thanks for the thorough explanation :-)

(https://discuss.leetcode.com/user/midgar77)

tusharbisht commented 2 months ago

Great article! Thanks!

(https://discuss.leetcode.com/user/tusharbisht)

lahiru commented 2 months ago

@awice (https://discuss.leetcode.com/uid/71269) Sure I get it. Thanks a lot.

(https://discuss.leetcode.com/user/lahiru)

awice commented 2 months ago

@chd (https://discuss.leetcode.com/uid/290175) I corrected it to better explain.

@lahiru (https://discuss.leetcode.com/uid/276991) The only times that use digits from {2, 3, 5, 9} are 22:22 and 23:59. After 23:59 (so when it's midnight), 22:22 will occur before 23:59, so it is the answer.

(https://discuss.leetcode.com/user/awice)

lahiru commented 2 months ago

I get output for 23:59 as 22:22 which is wrong.

(https://discuss.leetcode.com/user/lahiru)

chd commented 3 months ago

For the second method, I am still confused about the sentence "The best possible time != start is the one with the largest (time - start) % (24 * 60)", why this best possible is the closet time?

(https://discuss.leetcode.com/user/chd)

View original thread (https://discuss.leetcode.com/topic/105210)