**Next Closest Time**

Question

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:** time = "19:34"

**Output:** "19:39"

**Explanation:** The next closest time choosing from digits **1**, **9**, **3**, **4**, is **19:39**, which occurs 5 minutes later.

It is not **19:33**, because this occurs 23 hours and 59 minutes later.

**Example 2:**

**Input:** time = "23:59"

**Output:** "22:22"

**Explanation:** The next closest time choosing from digits **2**, **3**, **5**, **9**, is **22:22**.

It may be assumed that the returned time is next day's time since it is smaller than the input time numerically.

**Constraints:**

* time.length == 5
* time is a valid time in the form "HH:MM".
* 0 <= HH < 24
* 0 <= MM < 60

# **Solution**

#### **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.

Coding Solution

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);  }  }  }  } |

Python

|  |
| --- |
| 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)*O*(1). We try up to 24 \* 6024∗60 possible times until we find the correct time.
* Space Complexity: O(1)*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 minutes 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.

Coding Solution

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);  }  } |

Python

|  |
| --- |
| 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)*O*(1). We all 4^444 possible times and take the best one.
* Space Complexity: O(1)*O*(1).