# Next Closest Time

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 o
ccurs 5 minutes later.  It is not 19:33, because this occurs 23 hours and 59 minutes
later.
```

## 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 that the returned time is next day's time since it is smaller than the inp
ut time numerically.
```

### Solution 1

Just turn the clock forwards one minute at a time until you reach a time with the original digits.

```
from datetime import *

class Solution(object):
    def nextClosestTime(self, time):
        digits = set(time)
        while True:
            time = (datetime.strptime(time, '%H:%M') + timedelta(minutes=1)).strfti
me('%H:%M')
            if set(time) <= digits:
                return time
```

### Solution 2

Return the smallest time that uses the given digits, just make being larger than the input a priority.

```
def nextClosestTime(self, time):
    return min((t <= time, t)
               for i in range(24 * 60)
               for t in ['%02d:%02d' % divmod(i, 60)]
               if set(t) <= set(time))[1]
```

Golfed:

```
def nextClosestTime(self, T):
    return min((t<=T,t)for i in range(1440)for t in['%02d:%02d'%divmod(i,60)]if set
(t)<=set(T))[1]
```

written by StefanPochmann original link here

## Solution 2

Since there are at max `4 * 4 * 4 * 4` = `256` possible times, just try them all...

```
class Solution {
    int diff = Integer.MAX_VALUE;
    String result = "";

    public String nextClosestTime(String time) {
        Set<Integer> set = new HashSet<>();
        set.add(Integer.parseInt(time.substring(0, 1)));
        set.add(Integer.parseInt(time.substring(1, 2)));
        set.add(Integer.parseInt(time.substring(3, 4)));
        set.add(Integer.parseInt(time.substring(4, 5)));

        if (set.size() == 1) return time;

        List<Integer> digits = new ArrayList<>(set);
        int minute = Integer.parseInt(time.substring(0, 2)) * 60 + Integer.parseInt
(time.substring(3, 5));

        dfs(digits, "", 0, minute);

        return result;
    }

    private void dfs(List<Integer> digits, String cur, int pos, int target) {
        if (pos == 4) {
            int m = Integer.parseInt(cur.substring(0, 2)) * 60 + Integer.parseInt(c
ur.substring(2, 4));
            if (m == target) return;
            int d = m - target > 0 ? m - target : 1440 + m - target;
            if (d < diff) {
                diff = d;
                result = cur.substring(0, 2) + ":" + cur.substring(2, 4);
            }
            return;
        }

        for (int i = 0; i < digits.size(); i++) {
            if (pos == 0 && digits.get(i) > 2) continue;
            if (pos == 1 && Integer.parseInt(cur) * 10 + digits.get(i) > 23) contin
ue;
            if (pos == 2 && digits.get(i) > 5) continue;
            if (pos == 3 && Integer.parseInt(cur.substring(2)) * 10 + digits.get(i)
> 59) continue;
            dfs(digits, cur + digits.get(i), pos + 1, target);
        }
    }
}
```

written by shawngao original link here

## Solution 3

Solution 1:
we can try to increase the minute and the hour one by one. If all these four digits of the next time is in hashset, we find it and output! because these four digits are all reused.

```java
    public String nextClosestTime(String time) {
        String[] val = time.split(":");
        Set<Integer> set = new HashSet<>();
        int hour = add(set, val[0]);
        int minu = add(set, val[1]);

        int[] times = new int[] {hour, minu};
        nxt(times);

        while (!contains(times[0], times[1], set)) {
            nxt(times);
        }
        return valid(times[0]) + ":" + valid(times[1]);
    }

    public void nxt(int[] time) {
        int hour = time[0];
        int minu = time[1];
        minu ++;
        if (minu == 60) {
            hour ++;
            minu = 0;
            if (hour == 24) hour = 0;
        }
        time[0] = hour;
        time[1] = minu;
    }

    public int add(Set<Integer> set, String timeStr) {
        int time = Integer.parseInt(timeStr);
        set.add(time / 10);
        set.add(time % 10);
        return time;
    }

    public String valid(int time) {
        if (time >= 0 && time <= 9) return "0" + time;
        else return time + "";
    }

    public boolean contains(int hour, int minu, Set<Integer> set) {
        return set.contains(hour / 10) && set.contains(hour % 10) && set.contains(m
inu / 10) && set.contains(minu % 10);
    }
```

but in this way, the cost will be 24* 60 = 1440.

Solution 2:
Because these four digits can be reused,so the next time can be constructed by these

digits, so we try to use dfs to search all the next time.

it will search 4 * 4 * 4 * 4 = 256 times.

```java
    int diff = 0x3f3f3f3f;
    String result = "";
    int h;
    int m;
    public String nextClosestTime(String time) {
        int[] digit = new int[4];
        int tot = 0;
        String[] val = time.split(":");
        int hour = Integer.parseInt(val[0]);
        int minu = Integer.parseInt(val[1]);
        digit[tot++] = hour / 10;
        digit[tot++] = hour % 10;
        digit[tot++] = minu / 10;
        digit[tot++] = minu % 10;

        h = hour;
        m = minu;

        dfs(digit, 0, new int[4]);

        return result;
    }

    void dfs(int[] digit, int i, int[] ans) {
        if (i == 4) {
            int hour = 10 * ans[0] + ans[1];
            int minu = 10 * ans[2] + ans[3];
            if (hour >= 0 && hour <= 23 && minu >= 0 && minu <= 59) {
                int df = diff(hour, minu);
                if (df < diff) {
                    diff = df;
                    result = valid(hour) + ":" + valid(minu);
                }
            }
        }
        else {
            for (int j = 0; j < 4; ++j) {
                ans[i] = digit[j];
                dfs(digit, i + 1, ans);
            }
        }
    }

    int diff(int hour, int minu) {
        int c2o = 60 * 60 - h * 60 - m;
        int n2o = 60 * 60 - hour * 60 - minu;
        return n2o < c2o ? c2o - n2o : c2o - n2o + 3600;
    }

    public String valid(int time) {
        if (time >= 0 && time <= 9) return "0" + time;
        else return time + "";
    }
```

Solution 3:
Of course, we can try to prune, if hour and minute is not valid, just stop search.

for the test case "23:59", it will search 24 times, but solution 2 searches 256 times.

```
    int diff = 0x3f3f3f3f;
    String result = "";
    int h;
    int m;
    public String nextClosestTime(String time) {
        int[] digit = new int[4];
        int tot = 0;
        String[] val = time.split(":");
        int hour = Integer.parseInt(val[0]);
        int minu = Integer.parseInt(val[1]);
        digit[tot++] = hour / 10;
        digit[tot++] = hour % 10;
        digit[tot++] = minu / 10;
        digit[tot++] = minu % 10;

        h = hour;
        m = minu;

        dfs(digit, 0, new int[4]);

        return result;
    }

    void dfs(int[] digit, int i, int[] ans) {
        if (i == 4) {
            int hour = 10 * ans[0] + ans[1];
            int minu = 10 * ans[2] + ans[3];
            int df = diff(hour, minu);
            if (df < diff) {
                diff = df;
                result = valid(hour) + ":" + valid(minu);
            }
        }
        else {
            for (int j = 0; j < 4; ++j) {
                ans[i] = digit[j];
                if (i == 1) {
                    int hour = 10 * ans[0] + ans[1];
                    if (hour >= 0 && hour <= 23) dfs(digit, i + 1, ans);
                }
                else if (i == 3) {
                    int minu = 10 * ans[2] + ans[3];
                    if (minu >= 0 && minu <= 59) dfs(digit, i + 1, ans);
                }
                else {
                    dfs(digit, i + 1, ans);
                }
            }
        }
    }

    int diff(int hour, int minu) {
        int c2o = 60 * 60 - h * 60 - m;
        int n2o = 60 * 60 - hour * 60 - minu;
        return n2o < c2o ? c2o - n2o : c2o - n2o + 3600;
    }

    public String valid(int time) {
        if (time >= 0 && time <= 9) return "0" + time;
        else return time + "";
    }
```

written by DemonSong original link here