题目描述

警察在侦破一个案件时，得到了线人给出的可能犯罪时间，形如 "HH:MM" 表示的时刻。

根据警察和线人的约定，为了隐藏，该时间是经改过的。

解密规则为：利用当前出现过的数字，构造下一个距离当前时间最近的时刻，则该时间为可能的犯罪时间。

每个出现数字都可以被无限次使用。

输入描述

形如HH:SS字符串，表示原始输入。

输出描述

形如HH:SS的字符串，表示重组上限的犯罪时间。

备注

1.可以保证给出的时间字符串一定是合法的。

例如，"01:26"和"11:08"是合法的，"1:35"和" 11:8"是不合法的。

2.最近的时间可能在第二天。

用例

输入	输出
20:12	20:20
23:59	22:22
12:56	15:11
18:52	18:55
23:52	23:53
09:17	09:19
07:08	08:00

题目解析

> 解题思路为：利用当前出现过的数字，构造**下一个**距离当前时间最近的时刻，则该时间为可能的犯罪时间。

本题重点理解**下一个**最近时间，这里是“下一个”，而不是“上一个”，因此要求的时间在当前时间之后。但是题主中又说：“最近的时间可能在第二天。”

也挺是说，要求的时间 并不是严格 比当前时间 小的。

这题其实用dfs来**全排列**？的话，就很简单。

我们基于dfs来得输入“20:12”的全部排列。

则由0，1，2构成的四组排列，最后由“每个出现数字都可以被无限次使用”，因此每组多可以使用相同的数字。

当然，非常的全排列中必然含有不合法的时间，我们需要利用正则来过滤掉非法的时间，正则如下

```
const regExp = /([01][0-9]|(2[0-3]))(0-5][0-9]);
```

剩余的全排列如下

```
[
  "2222", "2220", "2221", "2202", "2200", "2203",
  "2212", "2210", "2211", "2002", "2000", "2003",
  "2002", "2000", "2001", "2012", "2010", "2011",
  "2122", "2120", "2121", "2202", "2200", "2201",
  "2222", "2220", "2221", "0222", "0220", "0223",
  "0202", "0200", "0203", "0212", "0210", "0213",
  "0022", "0020", "0023", "0002", "0000", "0003",
  "0012", "0010", "0013", "0122", "0120", "0123",
  "0102", "0100", "0103", "0112", "0110", "0113",
  "1222", "1220", "1221", "1202", "1200", "1203",
  "1212", "1210", "1213", "1022", "1020", "1023",
  "1002", "1000", "1003", "1012", "1010", "1013",
  "1122", "1120", "1121", "1102", "1100", "1103",
  "1112", "1110", "1111"
]
```

我们将其按字典序升序

```
[
  "0000", "0001", "0002", "0010", "0011", "0012",
  "0020", "0021", "0022", "0100", "0101", "0102",
  "0110", "0111", "0112", "0120", "0121", "0122",
  "0200", "0201", "0202", "0210", "0211", "0212",
  "0220", "0221", "0222", "1000", "1001", "1002",
  "1010", "1011", "1012", "1020", "1021", "1022",
  "1100", "1101", "1102", "1110", "1111", "1112",
  "1120", "1121", "1122", "1200", "1201", "1202",
  "1210", "1211", "1212", "1220", "1221", "1222",
  "2000", "2001", "2002", "2010", "2011", "2012",
  "2020", "2021", "2022", "2100", "2101", "2102",
  "2110", "2111", "2112", "2120", "2121", "2122",
  "2200", "2201", "2202", "2210", "2211", "2212",
  "2220", "2221", "2222"
]
```

找原输入的20:12的后20:12之后的一个时间2020，就是所求的。我们只要给2020中找出一个^^.就是答案。

另外如果输入的时间大于给定的时间字符串的时是排序后全排列最后一个，则此时我们需要取第二天的第一个时间，即排序后全排列第一个。

Java算法源码

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Scanner;
import java.util.regex.Pattern;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    String[] tmp = sc.nextLine().split(":");
    String hour = tmp[0];
    String minute = tmp[1];

    System.out.println(getResult(hour, minute));
  }

  public static String getResult(String hour, String minute) {
    HashSet<Character> set = new HashSet<>();
    for (int i = 0; i < hour.length(); i++) set.add(hour.charAt(i));
    for (int i = 0; i < minute.length(); i++) set.add(minute.charAt(i));
    Character[] arr = set.toArray(new Character[0]);

    ArrayList<String> res = new ArrayList<>();
    dfs(arr, new LinkedList<>(), res);

    res.sort(String::compareTo);
    int index = res.indexOf(hour + minute);

    String recentTime;
    if (index == res.size() - 1) {
      recentTime = res.get(0);
    } else {
      recentTime = res.get(index + 1);
    }

    String[] split = recentTime.split("");
    split[2] += ":";
    return String.join("", split);
  }

  static Pattern p = Pattern.compile("([01][0-9])|(2[0-3]))(0-5][0-9]");

  public static void dfs(Character[] arr, LinkedList<Character> path, ArrayList<String> res) {
    if (path.size() == 4) {
      StringBuilder sb = new StringBuilder();
      for (Character c : path) sb.append(c);
      String timeStr = sb.toString();

      if (p.matcher(timeStr).find()) {
        res.add(timeStr);
      }

      return;
    }

    for (Character c : arr) {
      path.add(c);
      dfs(arr, path, res);
      path.removeLast();
    }
  }
}
```

JS算法源码

```
// Node.js(ES5) Web 浏览器(ES5) 交互式输入/输出 版 //
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const [hour, minute] = line.split(":");
  console.log(getResult(hour, minute));
});

function getResult(hour, minute) {
  const arr = [...new Set([...hour, ...minute])];

  const res = [];
  dfs(arr, [], res);
  res.sort();

  let index = res.indexOf(`${hour}${minute}`);

  let recentTime;
  if (index === res.length - 1) {
    recentTime = [...res[0]];
  } else {
    recentTime = [...res[index + 1]];
  }

  recentTime.splice(2, 0, ":");

  return recentTime.join("");
}

function dfs(arr, path, res) {
  if (path.length === 4) {
    const timeStr = path.join("");
    const regExp = /([01][0-9])|(2[0-3]))(0-5][0-9]);
    if (regExp.test(timeStr)) res.push(timeStr);
    return;
  }

  for (let i = 0; i < arr.length; i++) {
    path.push(arr[i]);
    dfs(arr, path, res);
    path.pop();
  }
}
```

Python算法源码

```
import re

rep = re.compile("([01][0-9])|(2[0-3]))(0-5][0-9]")

# 输入输出
hour, minute = input().split(":")


def dfs(arr, path, res):
    if len(path) == 4:
        timeStr = "".join(path)
        if rep.search(timeStr) is not None:
            res.append(timeStr)
        return

    for i in range(len(arr)):
        path.append(arr[i])
        dfs(arr, path, res)
        path.pop()

# 算法入口
def getResult():
    arr = list(hour)
    arr.extend(list(minute))

    arr = list(set(arr))

    res = []
    dfs(arr, [], res)
    res.sort()

    index = res.index(hour + minute)

    if index == len(res) - 1:
        recentTime = res[0]
    else:
        recentTime = res[index + 1]

    ans = list(recentTime)
    ans[2] += ":"
    return "".join(ans)

# 调用算法
print(getResult())
```