题目描述

给定一个数组，里面有 6 个整数，求这个数组能够表示的最大 24 进制的时间是多少，输出这个时间，无法表示输出 invalid。

输入描述

输入为一个整数数组，数组内有六个整数。

输入整数数组长度为 6，不需要考虑其它长度，元素值为 0 或者正整数，6 个数字每个数字只能使用一次。

输出描述

输出为一个 24 进制格式的时间，或者字符串"invalid"。

用例

输入	[0,2,3,0,5,6]
输出	23:56:00
说明	无

题目解析

本题可以使用 深度优先 搜索DFS求解全排列，当然求解过程中需要过滤掉不合法的时间排列，然后剩下只需要进行默认的字典序升序后，获取最后一个时间排列就是最大的时间

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

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

rl.on("line", (line) => {
  const arr = JSON.parse(line);

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

  let dfs = (arr, used, path, res) => {
    if (path.length === arr.length) {
      if (regExp.test(path.join(""))) res.push([...path]);
      return;
    }

    for (let i = 0; i < arr.length; i++) {
      if (!used[i]) {
        path.push(arr[i]);
        used[i] = true;
        dfs(arr, used, path, res);
        used[i] = false;
        path.pop();
      }
    }
  };

  const res = [];
  dfs(arr, arr.slice().fill(false), [], res);

  if (!res.length) return console.log("invalid");

  const max = res.sort().at(-1);
  console.log(`${max[0]}${max[1]}:${max[2]}${max[3]}:${max[4]}${max[5]}`);
});
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.Arrays;
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 s = sc.nextLine();
    Integer[] arr =
        Arrays.stream(s.substring(1, s.length() - 1).split(","))
            .map(Integer::parseInt)
            .toArray(Integer[]::new);

    System.out.println(getResult(arr));
  }

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

  public static String getResult(Integer[] arr) {
    ArrayList<String> res = new ArrayList<>();
    dfs(arr, new boolean[arr.length], new LinkedList<>(), res);

    if (res.size() == 0) return "invalid";
    res.sort((a, b) -> b.compareTo(a));
    return res.get(0);
  }

  public static void dfs(
      Integer[] arr, boolean[] used, LinkedList<Integer> path, ArrayList<String> res) {
    if (path.size() == arr.length) {
      Integer[] t = path.toArray(new Integer[0]);
      String time = t[0] + "" + t[1] + ":" + t[2] + "" + t[3] + ":" + t[4] + "" + t[5];

      if (c.matcher(time).matches()) res.add(time);
      return;
    }

    for (int i = 0; i < arr.length; i++) {
      if (!used[i]) {
        path.add(arr[i]);
        used[i] = true;
        dfs(arr, used, path, res);
        used[i] = false;
        path.removeLast();
      }
    }
  }
}
```

Python算法源码

```python
import re

# 输入获取
arr = eval(input())

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


def dfs(arr, used, path, res):
    if len(path) == len(arr):
        tmp = "".join(map(str, path))
        if p.match(tmp):
            res.append(path[:])
        return

    for i in range(len(arr)):
        if not used[i]:
            path.append(arr[i])
            used[i] = True
            dfs(arr, used, path, res)
            used[i] = False
            path.pop()


# 算法入口
def getResult():
    res = []
    dfs(arr, [False] * (len(arr)), [], res)

    if len(res) == 0:
        return "invalid"

    res.sort()
    t = res[-1]

    return f"{t[0]}{t[1]}:{t[2]}{t[3]}:{t[4]}{t[5]}"


# 算法调用
print(getResult())
```