题目描述

某个打印机根据打印队列执行打印任务。打印任务分为九个优先级，分别用数字1-9表示，数字越大优先级越高。打印机每次从队列头部取出第一个任务A，

然后检查队列余下任务中有没有比A优先级更高的任务，如果有比A优先级高的任务，则将任务A放到队列尾部，否则就执行任务A的打印。

请编写一个程序，根据输入的打印队列，输出**实际的打印顺序。**

输入描述

输入一行，为每个任务的优先级，优先级之间用逗号隔开，优先级取值范围是1~9。

输出描述

输出一行，为每个任务的打印顺序，打印顺序从0开始，用逗号隔开

用例

输入	9,3,5
输出	0,2,1
说明	队列头部任务的优先级为9，最先打印，故序号为0； 接着队列头部任务优先级为3，队列中还有优先级为5的任务，优先级3任务被移到队列尾部； 接着打印优先级为5的任务，故其序号为1； 最后优先级为3的任务的序号为2。

输入	1,2,2
输出	2,0,1
说明	队列头部任务的优先级为1，被移到队列尾部；接着顺序打印两个优先级为2的任务，故其序号分别为0和1；最后打印剩下的优先级为1的任务，其序号为2

题目解析

简单的排序问题。

以用例1为例，假设初始时每个任务的索引，是其初始序号，比如9,3,5中：

- 9的初始序号是0
- 3的初始序号是1
- 5的初始序号是2

接下来，每次出队的其实都是优先级最高的，即我们只需要将9,3,5按照优先级进行降序排序，即为任务出队的顺序，即降序后为9,5,3：

- 9的出队序号是0
- 5的出队序号是1
- 3的出队序号是2

然后我们需要按照初始序号的顺序来打印出队序号，即

- 初始序号0，对应9，对应的出队序号是0
- 初始序号1，对应3，对应的出队序号是2
- 初始序号2，对应2，对应的出队序号是1

我的解题思路如下，首先将输入的数组[9,3,5] 映射为值-索引对，即转化为[[9,0], [3,1], [5,2]]

然后按照优先级排序，得到[[9,0], [5,2], [3,1]]，此时该数组的索引-值对应关系如下：

- 0：[9, 0]
- 1：[5, 2]
- 2：[3, 1]

含义是

> 出队序号：[优先级，初始序号]

然后定义一个ids数组，ids[初始序号] = 出队序号

- ids[0] = 0
- ids[1] = 2
- ids[2] = 1

最后打印ids数组即可。

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;
import java.util.StringJoiner;

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

    Integer[] priority =
        Arrays.stream(sc.nextLine().split(",")).map(Integer::parseInt).toArray(Integer[]::new);

    int n = priority.length;

    int[][] tasks = new int[n][2];
    for (int i = 0; i < n; i++) {
      tasks[i] = new int[] {priority[i], i}; // 优先级，初始序号
    }

    Arrays.sort(tasks, (a, b) -> b[0] - a[0]); // 按照优先级排序

    int[] ids = new int[n];
    for (int i = 0; i < n; i++) {
      ids[tasks[i][1]] = i; // 将 排序序号 对应到 初始序号上
    }

    StringJoiner sj = new StringJoiner(",");
    for (int id : ids) {
      sj.add(id + "");
    }
    System.out.println(sj);
  }
}
```

JS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 tasks = line
    .split(",")
    .map((priority, index) => [priority, index]) // 优先级, 初始序号
    .sort((a, b) => b[0] - a[0]); // 按照优先级降序排序

  const n = tasks.length;

  const ids = new Array(n);
  for (let i = 0; i < n; i++) {
    ids[tasks[i][1]] = i; // 将 排序后序号 对应到 初始序号上
  }

  console.log(ids.join(","));
});
```

Python算法源码

```python
# 输入获取
arr = list(map(int, input().split(",")))


# 算法入口
def getResult():
    n = len(arr)

    tasks = []
    for i in range(n):
        priority = arr[i]
        tasks.append((priority, i))

    tasks.sort(key=lambda x: -x[0])

    ids = [0] * n
    for i in range(n):
        ids[tasks[i][1]] = i

    return ",".join(map(str, ids))


# 算法调用
print(getResult())
```