题目描述

有5台打印机打印文件，每台打印机有自己的待打印队列。

因为打印的文件内容有轻重缓急之分，所以队列中的文件有1~10不同的代先级，其中**数字越大优先级越高**。

打印机会从自己的待打印队列中选择***优先级最高***的文件来打印。

如果存在两个优先级一样的文件，则选择***最早进入队列***的那个文件。

现在请你来模拟这5台打印机的打印过程。

输入描述

每个输入包含1个测试用例，

每个测试用例第一行给出发生事件的数量N（0 < N < 1000）。

接下来有 N 行，分别表示发生的事件。共有如下两种事件：

1. “IN P NUM”，表示有一个拥有优先级 NUM 的文件放到了打印机 P 的待打印队列中。（0< P <= 5, 0 < NUM <= 10）；
2. “OUT P”，表示打印机 P 进行了一次文件打印，同时该文件从待打印队列中取出。（0 < P <= 5）。

输出描述

- 对于每个测试用例，每次”OUT P”事件，请在一行中**输出文件的编号**。
- 如果此时没有文件可以打印，请输出”**NULL**“。
- 文件的编号定义为”IN P NUM”事件发生第 x 次，此处待打印文件的编号为x。编号从**1**开始。

用例

输入	7 IN 1 1 IN 1 2 IN 1 3 IN 2 1 OUT 1 OUT 2 OUT 2
输出	3 4 NULL
说明	无

输入	5 IN 1 1 IN 1 3 IN 1 1 IN 1 3 OUT 1
输出	2
说明	无

题目解析

本题可以基于 优先队列 实现打印机总是打印优先级最高的文件。

优先队列，如果想简单一点的话，则可以基于 有序数组 实现，但是有序数组是整体有序，每次有新任务入队，都需要O(n)时间复杂度维持。

优先队列最好是基于堆结构实现，所谓堆结构，即一颗 完全二叉树 。本题是优先级数值越大，优先级越高，因此我们可以使用大顶堆。

关于基于堆结构实现优先队列，可以参考

题目解析

本题可以基于 优先队列 实现打印机总是打印优先级最高的文件。

优先队列，如果想简单一点的话，则可以基于 有序数组 实现，但是有序数组是整体有序，每次有新任务入队，都需要O(n)时间复杂度维持。

优先队列最好是基于堆结构实现，所谓堆结构，即一颗 完全二叉树 。本题是优先级数值越大，优先级越高，因此我们可以使用大顶堆。

关于基于堆结构实现优先队列，可以参考

LeetCode - 1705 吃苹果的最大数目_伏城之外的博客-CSDN博客

JavaScript算法源码

基于有序数组实现优先队列

```
/* 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,
});

const lines = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    n = parseInt(lines[0]);
  }

  if (n && lines.length === n + 1) {
    lines.shift();

    const matrix = lines.map((line) => line.split(" "));

    getResult(matrix);

    lines.length = 0;
  }
});

function getResult(matrix) {
  const print = {};

  let taskId = 1;
  matrix.forEach((task) => {
    const [type, printId, priority] = task;

    if (type === "IN") {
      const arr = [taskId, priority];
      if (!print[printId]) {
        print[printId] = []; // 基于数组实现优先队列
      }
      print[printId].push(arr);
      print[printId].sort((a, b) => b[1] - a[1]); // 维持高优先级在头部
      taskId++;
    } else {
      const arr = print[printId].shift();
      console.log(arr ? arr[0] : "NULL");
    }
  });
}
```

基于堆结构实现优先队列

```
/* 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,
});

const lines = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    n = parseInt(lines[0]);
  }

  if (n && lines.length === n + 1) {
    lines.shift();

    const matrix = lines.map((line) => line.split(" "));

    getResult(matrix);
```

基于堆结构实现优先队列

```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,
});

const lines = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    n = parseInt(lines[0]);
  }

  if (n && lines.length === n + 1) {
    lines.shift();

    const matrix = lines.map((line) => line.split(" "));

    getResult(matrix);

    lines.length = 0;
  }
});

function getResult(matrix) {
  const print = {};

  let taskId = 1;
  matrix.forEach((task) => {
    const [type, printId, priority] = task;

    if (type === "IN") {
      const arr = [taskId, priority];
      if (!print[printId]) {
        print[printId] = new PriorityQueue((a, b) => a[1] - b[1]);// 基于大顶堆实现优先队列
      }
      print[printId].push(arr);
      taskId++;
    } else {
      const arr = print[printId].shift();
      console.log(arr ? arr[0] : "NULL");
    }
  });
}

class PriorityQueue {
  constructor(cpr) {
    this.queue = [];
    this.cpr = cpr;
  }

  swap(a, b) {
    const tmp = this.queue[a];
    this.queue[a] = this.queue[b];
    this.queue[b] = tmp;
  }

  // 上浮
  swim() {
    let c = this.queue.length - 1;

    while (c >= 1) {
      const f = Math.floor((c - 1) / 2);

      if (this.cpr(this.queue[c], this.queue[f]) > 0) {
        this.swap(c, f);
        c = f;
      } else {
        break;
      }
    }
  }

  // 入队
  push(val) {
    this.queue.push(val);
    this.swim();
  }

  // 下沉
  sink() {
    let f = 0;

    while (true) {
      let c1 = 2 * f + 1;
      let c2 = c1 + 1;

      let c;
```

```
  // 入队
  push(val) {
    this.queue.push(val);
    this.swim();
  }

  // 下沉
  sink() {
    let f = 0;

    while (true) {
      let c1 = 2 * f + 1;
      let c2 = c1 + 1;

      let c;
      let val1 = this.queue[c1];
      let val2 = this.queue[c2];
      if (val1 && val2) {
        c = this.cpr(val1, val2) > 0 ? c1 : c2;
      } else if (val1 && !val2) {
        c = c1;
      } else if (!val1 && val2) {
        c = c2;
      } else {
        break;
      }

      if (this.cpr(this.queue[c], this.queue[f]) > 0) {
        this.swap(c, f);
        f = c;
      } else {
        break;
      }
    }
  }

  // 出队
  shift() {
    this.swap(0, this.queue.length - 1);
    const res = this.queue.pop();
    this.sink();
    return res;
  }
}
```

Java算法源码

Java已经有优先队列实现类PriorityQueue，因此可以直接使用它。

```
import java.util.HashMap;
import java.util.PriorityQueue;
import java.util.Scanner;

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

    int n = Integer.parseInt(sc.nextLine());
    String[][] matrix = new String[n][];

    for (int i = 0; i < n; i++) {
      String[] s = sc.nextLine().split(" ");
      matrix[i] = s;
    }

    getResult(matrix);
  }

  public static void getResult(String[][] matrix) {
    // print中存放每台打印机的等待队列
    HashMap<String, PriorityQueue<Integer[]>> print = new HashMap<>();

    // 文件的编号定义为"IN P NUM"事件发生第 x 次，此处待打印文件的编号为x。编号从1开始。
    int x = 1;
    for (String[] task : matrix) {
      // IN,OUT都有type和printId
      String type = task[0];
      String printId = task[1];

      if ("IN".equals(type)) {
        // IN还有priority
        String priority = task[2];
        // arr是打印任务
        Integer[] arr = {x, Integer.parseInt(priority)};
        // 为打印机printId设置打印优先级，打印任务的priority越大，优先级越高
        print.putIfAbsent(printId, new PriorityQueue<>((a, b) -> b[1] - a[1]));
        // 将打印任务加入对应打印机
        print.get(printId).offer(arr);
        x++;
      } else {
        // 打印机等待队列中取出优先级最高的打印任务arr
        Integer[] arr = print.get(printId).poll();
        if (arr != null) {
```

Java算法源码

Java已经有优先队列实现类PriorityQueue，因此可以直接使用它。

```java
import java.util.HashMap;
import java.util.PriorityQueue;
import java.util.Scanner;

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

    int n = Integer.parseInt(sc.nextLine());
    String[][] matrix = new String[n][];

    for (int i = 0; i < n; i++) {
      String[] s = sc.nextLine().split(" ");
      matrix[i] = s;
    }

    getResult(matrix);
  }

  public static void getResult(String[][] matrix) {
    // print中存放每台打印机的等待队列
    HashMap<String, PriorityQueue<Integer[]>> print = new HashMap<>();

    // 文件的编号定义为"IN P NUM"事件发生第 x 次，此处待打印文件的编号为x，编号从1开始。
    int x = 1;
    for (String[] task : matrix) {
      // IN,OUT都有type和printId
      String type = task[0];
      String printId = task[1];

      if ("IN".equals(type)) {
        // IN还有priority
        String priority = task[2];
        // arr是打印任务
        Integer[] arr = {x, Integer.parseInt(priority)};
        // 为打印机printId设置打印优先级，打印任务的priority越大，优先级越高
        print.putIfAbsent(printId, new PriorityQueue<>((a, b) -> b[1] - a[1]));
        // 将打印任务加入对应打印机
        print.get(printId).offer(arr);
        x++;
      } else {
        // 打印机等待队列中取出优先级最高的打印任务arr
        Integer[] arr = print.get(printId).poll();
        if (arr != null) {
          // arr[0]是x
          System.out.println(arr[0]);
        } else {
          // 如果此时没有文件可以打印，请输出"NULL"。
          System.out.println("NULL");
        }
      }
    }
  }
}
```

Python算法源码

```python
import queue

# 输入获取
n = int(input())

matrix = []
for i in range(n):
    matrix.append(input().split())


# 算法入口
def getResult(matrix):
    printer = {}

    taskId = 1
    for task in matrix:
        type = task[0]
        printerId = task[1]

        if type == "IN":
            priority = task[2]
            arr = (-int(priority), taskId)
            if printer.get(printerId) is None:
                printer[printerId] = queue.PriorityQueue()
            printer[printerId].put(arr)
            taskId += 1
        else:
            if printer.get(printerId) is None or printer[printerId].qsize() == 0:
                print("NULL")
            else:
                arr = printer[printerId].get()
```

Python算法源码

```python
import queue

# 输入获取
n = int(input())

matrix = []
for i in range(n):
    matrix.append(input().split())


# 算法入口
def getResult(matrix):
    printer = {}

    taskId = 1
    for task in matrix:
        type = task[0]
        printerId = task[1]

        if type == "IN":
            priority = task[2]
            arr = (-int(priority), taskId)
            if printer.get(printerId) is None:
                printer[printerId] = queue.PriorityQueue()
            printer[printerId].put(arr)
            taskId += 1
        else:
            if printer.get(printerId) is None or printer[printerId].qsize() == 0:
                print("NULL")
            else:
                arr = printer[printerId].get()
                print(arr[1])


getResult(matrix)
```