题目描述

给定一个正整数数组表示待系统执行的任务列表，数组的每一个元素代表一个任务，元素的值表示该任务的类型。

请计算执行完所有任务所需的最短时间。

任务执行规则如下:

1. 任务可以按任意顺序执行，且每个任务执行耗时间均为1个时间单位。
2. 两个同类型的任务之间必须有长度为N个单位的冷却时间，比如N为2时，在时间K执行了类型3的任务，那么K+1和K+2两个时间不能执行类型3任务。
3. 系统在任何一个单位时间内都可以执行一个任务，或者等待状态。

说明：数组最大长度为1000，速度最大值1000。

输入描述

- 第一行记录一个用半角逗号分隔的数组，数组长度不超过1000，数组元素的值不超过1000，
- 第二行记录任务冷却时间，N为正整数，N<=100。

输出描述

- 输出为执行完所有任务所需的最短时间。

用例

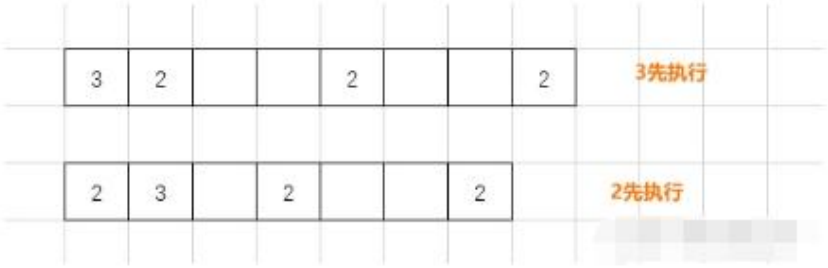
输入	2,2,2,3 2
输出	7
说明	时间1：执行类型2任务。 时间2：执行类型3的任务（因为冷却时间为2，所以时间2不能执行类型2的任务）。 时间3：系统等待（仍然在类型2的冷却时间）。 时间4：执行类型2任务。 时间5：系统等待。 时间6：系统等待。 时间7：执行类型2任务。 因此总共耗时7。

题目解析

本题需要使用贪心思维去解题。

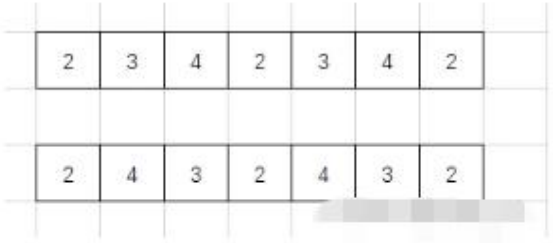
想要总的执行时间最短，即尽量保证每一单位时间都有任务被执行，避免空转等待。

比如用例1的执行策略有如下两种：



从上面策略可以看出，应该优先执行任务量多的某个任务。其他任务可以在它的冷却时间内执行。

再比如：2,2,2,3,3,4,4



因此，我们可以先统计出各种类型任务的数量，然后按照任务数量将任务降序排序

比如 2,2,2,3 统计为 [[2,3], [3,1]]，即任务2有3个，任务3有1个

我们可以再在统计结果上追加一个冷却时间，初始时冷却时间为0

tasks = [[2,3,0], [3,1,0]] 即：tasks = [[任务类型，数量，冷却时间]]。

统计过程中，我们还可以记录下所有的任务数量total = 4

定义一个time=0，记录时间。

接下来开始开始遍历tasks，time++

如果遍历到的tasks[i]的冷却时间为0，则将其数量--，即task[i][1]--，同时total--，另外该任务的冷却时间要变为第二行输入的wait时间，即task[i][2] = wait。

如果遍历到的task[i]的冷却时间不为0或者本轮已经有任务被执行了，则仅冷却时间--，即task[i][2]--，注意如果冷却时间已经为0，则放弃--。

检查total是否为0，如果不为0，则继续从头遍历tasks继续以上逻辑。

最后输出time作为题解。

我们演示下用例：2,2,2,3,3,4,4 的运行过程。

解析第一行输入得到arr：

```
arr = [2,2,2,3,3,4,4]
```

解析第二行输入得到wait：

```
wait = 2 // 假设第二行输入为2
```

统计不同类型的任务数量，并按任务数量降序排序，并初始化冷却时间为0，得到：

```
tasks = [[2,3,0], [3,2,0], [4,2,0]]  // [任务类型，数量，冷却时间]
```

统计任务总数：

```
total = arr.length
```

定义一个时间

```
time = 0
```

接下来多轮循环遍历tasks

```
while(total) // 只要还有任务等待执行
{
    time++
    let flag = true // 标记本轮时间是否可用
    for(let task of tasks) { // 遍历每个任务
        if(flag && task[1] > 0 && task[2] === 0) { // 本轮时间可用 && 有任务 && 任务冷却结束
            flag = false // 本轮时间已用
            task[1]-- // 完成一个任务，本任务数--
            total-- // 总任务数--
            task[2] = wait // 本任务进入冷却等待
        } else {
            task[2] > 0 ? task[2]-- : null
        }
    }
}
```

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,
});

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

  if (lines.length === 2) {
    const arr = lines[0].split(",").map(Number);
    const wait = lines[1] - 0;

    console.log(getResult(arr, wait));

    lines.length = 0;
  }
});

```

```javascript
function getResult(arr, wait) {
  const count = {};
  for (let t of arr) {
    count[t] ? count[t]++ : (count[t] = 1);
  }

  const tasks = [];
  for (let t in count) {
    tasks.push([count[t], 0]);
  }
  tasks.sort((a, b) => b[0] - a[0]);

  let total = arr.length;
  let time = 0;
  while (total) {
    time++;
    let flag = true;
    for (let i = 0; i < tasks.length; i++) {
      const task = tasks[i];
      if (flag && task[0] > 0 && task[1] === 0) {
        flag = false;
        task[0]--;
        total--;
        task[1] = wait;
      } else {
        task[1] > 0 ? task[1]-- : null;
      }
    }
  }

  return time;
}
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Scanner;

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

    Integer[] arr =
        Arrays.stream(sc.next().split(",")).map(Integer::parseInt).toArray(Integer[]::new);
    int wait = sc.nextInt();

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

  // 算法入口
  public static int getResult(Integer[] arr, int wait) {
    HashMap<Integer, Integer> count = new HashMap<>();

    for (Integer t : arr) {
      count.put(t, count.getOrDefault(t, 0) + 1);
    }

```

```
    ArrayList<Integer[]> tasks = new ArrayList<>();
    for (Integer t : count.keySet()) {
      tasks.add(new Integer[] {count.get(t), 0});
    }

    tasks.sort((a, b) -> b[0] - a[0]);

    int total = arr.length;
    int time = 0;

    while (total > 0) {
      time++;
      boolean flag = true;
      for (Integer[] task : tasks) {
        if (flag && task[0] > 0 && task[1] == 0) {
          flag = false;
          task[0]--;
          total--;
          task[1] = wait;
        } else {
          if (task[1] > 0) {
            task[1]--;
          }
        }
      }
    }

    return time;
  }
}
```

Python算法源码

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


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

    for t in arr:
        if count.get(t) is None:
            count[t] = 0

        count[t] += 1

```

```
    tasks = []
    for t in count.keys():
        tasks.append([count[t], 0])

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

    total = len(arr)
    time = 0

    while total > 0:
        time += 1
        flag = True
        for task in tasks:
            if flag and task[0] > 0 and task[1] == 0:
                flag = False
                task[0] -= 1
                total -= 1
                task[1] = wait
            else:
                if task[1] > 0:
                    task[1] -= 1

    return time


# 测试调用
print(getResult())
```