33、最少面试官数，考点 or 实现——贪心思维

题目描述

某公司组织一场公开招聘活动，假设由于人数和场地的限制，每人每次面试的时长不等，并已经安排给定，用(S1,E1)、(S2,E2)、(Sj,Ej)...(Si < Ei，均为非负整数)表示每场面试的开始和结束时间。

面试采用一对一的方式，即一名面试官同时只能面试一名应试者，一名面试官完成一次面试后可以立即进行下一场面试，且每个面试官的面试人次不超过 m。

为了支撑招聘活动高效顺利进行，请你计算至少需要多少名面试官。

输入描述

输入的第一行为面试官的最多面试人次 m，第二行为当天总的面试场次 n，

接下来的 n 行为每场面试的起始时间和结束时间，起始时间和结束时间用空格分隔。

其中， 1 <= n, m <= 500

输出描述

输出一个整数，表示至少需要的面试官数量。

用例

输入	2 5 1 2 2 3 3 4 4 5 5 6
输出	3
说明	总共有 5 场面试，且面试时间都不重叠，但每个面试官最多只能面试 2 人次，所以需要 3 名面试官。

题目解析

本题要想面试官人数最少，则需要每个面试官面试尽量多的人。

我们将每个面试官想象成一个桶，每个面试者想象成一个球。

首先，我们需要对面试者的面试时间段，按照结束时间进行升序。

由于不知道面试官个数，因此初始化时，有几个面试者就预定几个面试官，比如用例中有5个面试者，那就是初始化预定5个面试官。

接着，就是球放桶的问题了。

第一个球，放第一个桶。

第二个球，先尝试放第一个桶，首先检查桶中球个数是否已达到m，若已达到，则不能放入，继续尝试放入下一个桶，若未达到，则将球和桶最上面的球比较，即看面试时间是否有交集，如果有交集，则不能放入，继续尝试放入下一个桶，如果没有交集，则可以放入。

按照上面逻辑，放入所有球。

然后把空桶排除掉，剩下还有几个桶，就需要几个面试官。

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

  if (lines.length === 2) {
    m = lines[0] - 0;
    n = lines[1] - 0;
  }

  if (n && lines.length === n + 2) {
    const ranges = lines.slice(2).map((line) => line.split(" ").map(Number));
    console.log(getResult(ranges, m, n));
    lines.length = 0;
  }
});

function getResult(ranges, m, n) {
  // 求解最大不相交区间时，需要将所有区间按照右边界升序
  ranges.sort((a, b) => a[1] - b[1]);
  const buckets = new Array(n).fill(0).map(() => new Array());

  for (let [s, e] of ranges) {
    for (let bucket of buckets) {
      // 每个面试官最多面试m次
      if (bucket.length < m && (bucket.length == 0 || bucket.at(-1) < s)) {
        bucket.push(e);
        break;
      }
    }
  }

  return buckets.filter((bucket) => bucket.length > 0).length;
}
```

Java算法源码

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

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

    int[][] ranges = new int[n][2];
    for (int i = 0; i < n; i++) {
      ranges[i][0] = sc.nextInt();
      ranges[i][1] = sc.nextInt();
    }

    System.out.println(getResult(m, n, ranges));
  }

  // 算法入口
  public static long getResult(int m, int n, int[][] ranges) {
    // 求解最大不相交区间时，需要将所有区间按照右边界升序
    Arrays.sort(ranges, (a, b) -> a[1] - b[1]);

    ArrayList<LinkedList<Integer>> buckets = new ArrayList<>();
    for (int i = 0; i < n; i++) buckets.add(new LinkedList<>());

    for (int[] range : ranges) {
      int s = range[0];
      int e = range[1];

      for (LinkedList<Integer> bucket : buckets) {
        // 每个面试官最多面试m次
        if (bucket.size() < m && (bucket.size() == 0 || bucket.getLast() < s)) {
          bucket.add(e);
          break;
        }
      }
    }

    return buckets.stream().filter(bucket -> bucket.size() > 0).count();
  }
}
```

Python算法源码

```python
# 输入获取
m = int(input())
n = int(input())
areas = [list(map(int, input().split())) for _ in range(n)]


# 算法入口
def getResult():
    # 求解最大不相交区间时，需要将所有区间按照右边界升序
    areas.sort(key=lambda x: x[1])

    buckets = [[] for _ in range(n)]

    for s, e in areas:
        for bucket in buckets:
            # 每个面试官最多面试m次
            if len(bucket) < m and (len(bucket) == 0 or bucket[-1] < s):
                bucket.append(e)
                break

    return len(list(filter(lambda b: len(b) > 0, buckets)))


# 算法调用
print(getResult())
```