题目描述

给定一个二维整数矩阵，要在这个矩阵中选出一个子矩阵，使得这个子矩阵内所有的数字和尽量大，我们把这个子矩阵称为和最大子矩阵，子矩阵的选取原则是原矩阵中一块相互连续的矩形区域。

输入描述

输入的第一行包含2个整数n, m(1 <= n, m <= 10)，表示一个n行m列的矩阵，下面有n行，每行有m个整数，同一行中，每2个数字之间有1个空格，最后一个数字后面没有空格，所有的数字的在[-1000, 1000]之间。

输出描述

输出一行一个数字，表示选出的和最大子矩阵内所有的数字和。

用例

输入	3 4 -3 5 -1 5 2 4 -2 4 -1 3 -1 3
输出	20
说明	一个3*4的矩阵中，后面3列的子矩阵求和加起来等于20，和最大。

题目分析

这道题首先要考虑处理输入处理的问题，真的讨厌这种牛客式的编程模式，需要自己组装程序入参，还要处理多行输入，直接给个二维数组入参不行吗.....真的无语了，Leetcode就做的挺好的。

唉，抱怨归抱怨，下面是根据多行输入的信息，生成martrix二维数组的逻辑

```
/* 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,
});

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

  // 输入第一行时，提取出m、n
  if (lines.length === 1) {
    [n, m] = lines[0].split(" ").map((ele) => parseInt(ele));
  }

```

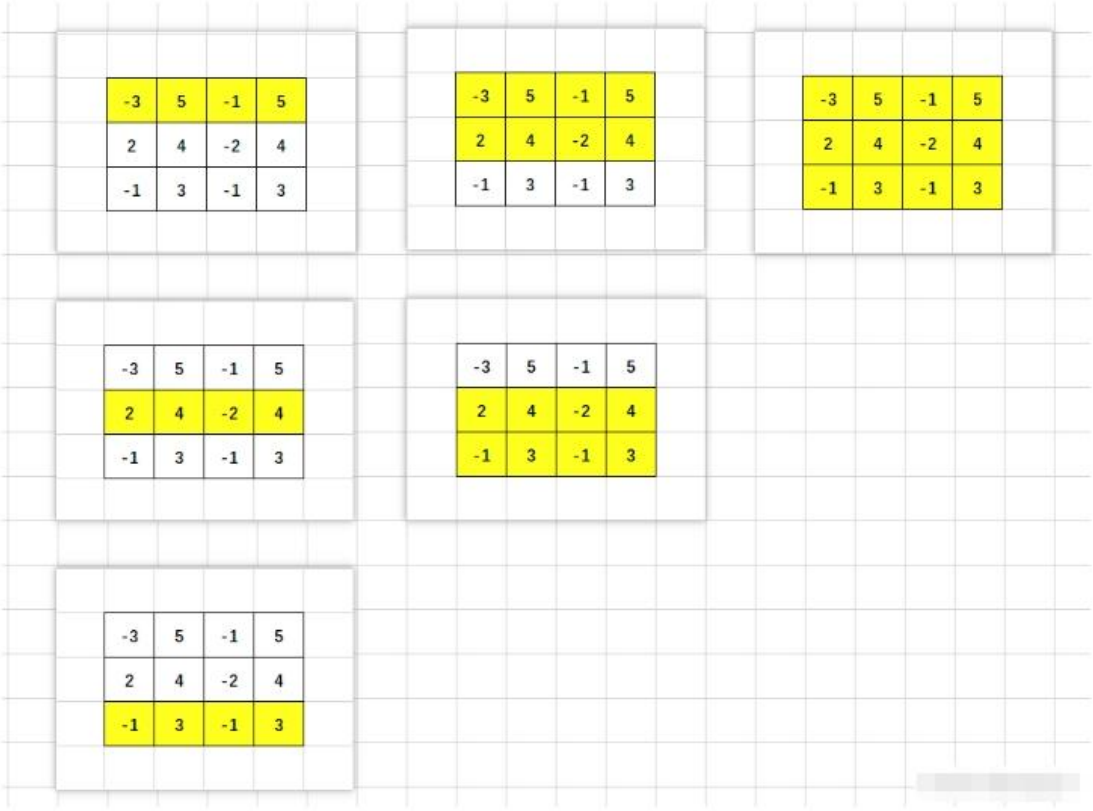
```
  // 输入第一行后，再输入n行时，则开始自动算法程序
  if (lines.length - 1 === n) {
    // 干掉第一行输入，则lines中存储的全是就是matrix要素
    lines.shift();

    // matrix是算法程序的入参二维数组
    let matrix = [];
    // 将多行输入的matrix要素提取出来存到二维数组中
    lines.forEach((line) => {
      matrix.push(
        line
          .split(" ")
          .map((ele) => parseInt(ele))
          .slice(0, m)
      );
    });

    // 调用算法程序
    maxSubMatrixSum(matrix);

    // 将输入归0，重新接收下一轮
    lines.length = 0;
  }
});

function maxSubMatrixSum(matrix) {
  console.log(JSON.stringify(matrix));
}
```

然后我们再来思考算法程序的编写。

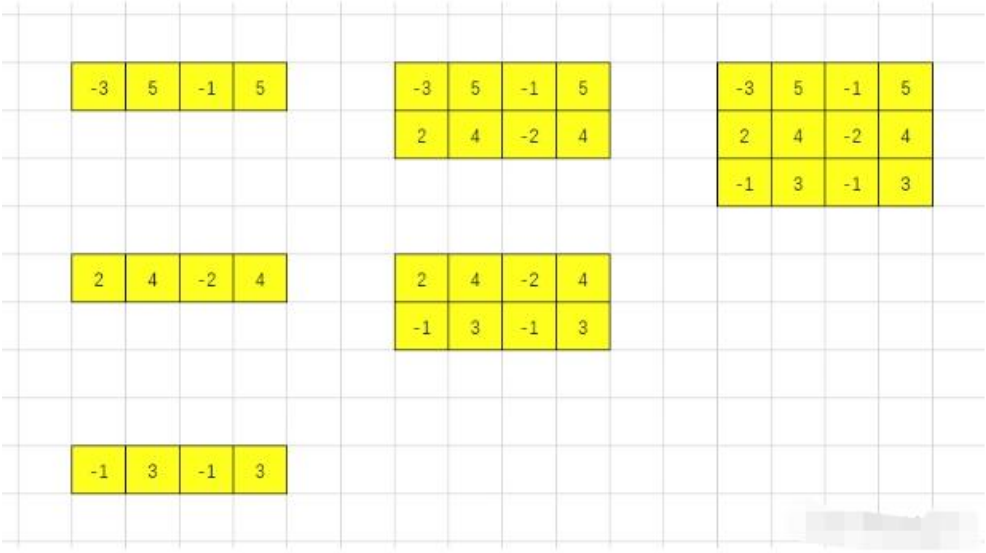
看到这个题目标题，我很容易就联想到了最大子数组和，区别在于最大子数组和是一维的，而最大子矩阵和是二维的。

那么是不是有可能将最大子矩阵和的求解转成一维的呢？

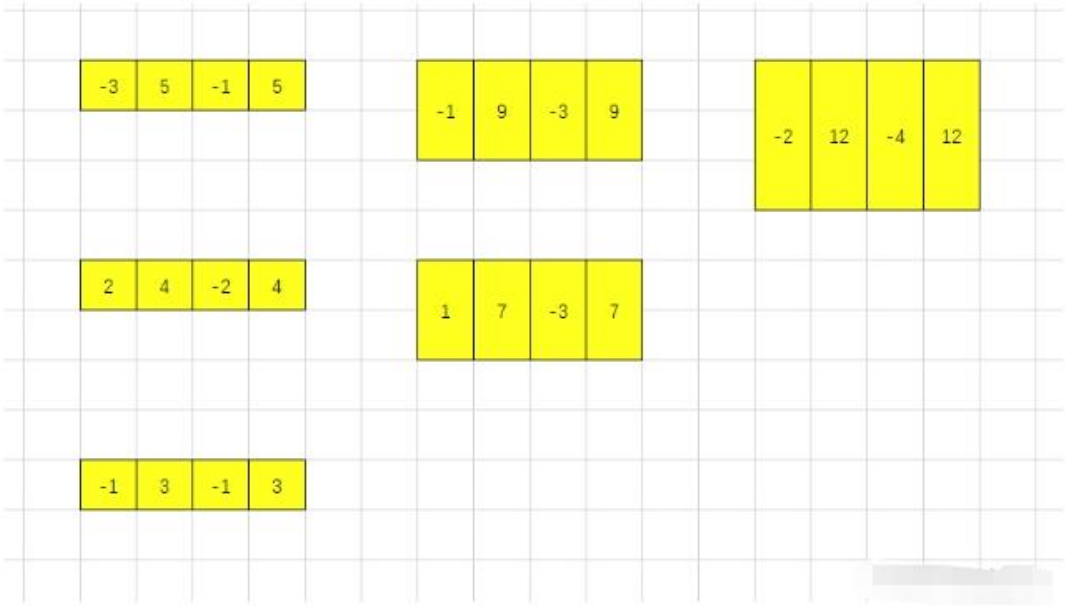
下面是子矩阵可能存在的区域，即一行子矩阵，两行子矩阵，三行子矩阵



进一步简化，可得下图，即对求解下面每个区域的最大子矩阵



此时因为子矩阵的行数已经确定，因此我们可以将多行压缩为一行



此时对于最大子矩阵和的求解，就变为了最大子数组和的求解。

而最大子数组和的求解的状态转移方程我们已经在前一小结总结出来了：

dp[i] = max(dp[i-1], 0) + nums[i]。

还有一个难点就是二维数组压缩为一维数组的问题，解决思路如下，获取二维数组的行数rows、列数cols，创建一个长度为cols的一维数组，然后将二维数组双重for循环，外层遍历cols，内层遍历rows，这样每次循环就可以得到二维数组一个列上的所有元素，然后求和存入一维数组中，实现如下

```
function matrixZip(matrix) {
  let cols = matrix[0].length;
  let rows = matrix.length;
  let zip = new Array(cols).fill(0);

  for (let c = 0; c < cols; c++) {
    for (let r = 0; r < rows; r++) {
      zip[c] += matrix[r][c];
    }
  }

  return zip;
}
```

JavaScript算法源码

```
/* 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,
});

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

  // 输入第一行时，提取出m、n
  if (lines.length === 1) {
    [n, m] = lines[0].split(" ").map((ele) => parseInt(ele));
  }

  // 输入第一行后，再输入n行时，则开始自动算法程序
  if (lines.length - 1 === n) {
    // 干掉第一行输入，即lines中存储的全是成员matrix要素
    lines.shift();

    // matrix是算法程序的入参二维数组
    let matrix = [];
    // 将多行输入的matrix要素提取出来存到二维数组中
    lines.forEach((line) => {
      matrix.push(
        line
          .split(" ")
          .map((ele) => parseInt(ele))
          .slice(0, m)
      );
    });

```

```
    // 调用算法程序
    console.log(maxSubMatrixSum(matrix));

    // 将输入归0，重新接收下一轮
    lines.length = 0;
  }
});

function maxSubMatrixSum(matrix) {
  let dp = [];
  for (let i = 0; i < matrix.length; i++) {
    dp.push(maxSubArraySum(matrix[i]));

    for (let j = i + 1; j < matrix.length; j++) {
      dp.push(maxSubArraySum(matrixZip(matrix.slice(i, j + 1))));
    }
  }

  return dp.sort((a, b) => b - a)[0];
}

function maxSubArraySum(nums) {
  let dp = new Array(nums.length);

  let result = (dp[0] = nums[0]);

  for (let i = 1; i < nums.length; i++) {
    dp[i] = Math.max(dp[i - 1], 0) + nums[i];
    result = Math.max(result, dp[i]);
  }

  return result;
}

function matrixZip(matrix) {
  let cols = matrix[0].length;
  let rows = matrix.length;
  let zip = new Array(cols).fill(0);

  for (let c = 0; c < cols; c++) {
    for (let r = 0; r < rows; r++) {
      zip[c] += matrix[r][c];
    }
  }

  return zip;
}
```

Java算法源码

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

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

    int n = sc.nextInt();
    int m = sc.nextInt();

    int[][] matrix = new int[n][m];
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        matrix[i][j] = sc.nextInt();
      }
    }

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

  public static int getResult(int n, int m, int[][] matrix) {
    ArrayList<Integer> dp = new ArrayList<>();

    for (int i = 0; i < n; i++) {
      dp.add(maxSubArraySum(matrix[i])); // 一行子矩阵最大和

      for (int j = i + 1; j < n; j++) {
        dp.add(maxSubArraySum(matrixZip(Arrays.copyOfRange(matrix, i, j + 1)))); // 多行子矩阵最大和
      }
    }

    return dp.stream().max((a, b) -> a - b).orElse(0); // 求出最大和
  }

```

```java
  // 最大子数组和求解
  public static int maxSubArraySum(int[] nums) {
    int[] dp = new int[nums.length];

    int res = dp[0] = nums[0];

    for (int i = 1; i < nums.length; i++) {
      dp[i] = Math.max(dp[i - 1], 0) + nums[i];
      res = Math.max(res, dp[i]);
    }

    return res;
  }

  // 多行子矩阵，压缩为一行子数组
  public static int[] matrixZip(int[][] matrix) {
    int cols = matrix[0].length;
    int rows = matrix.length;
    int[] zip = new int[cols];

    for (int c = 0; c < cols; c++) {
      for (int r = 0; r < rows; r++) {
        zip[c] += matrix[r][c];
      }
    }

    return zip;
  }
}
```

Python算法源码

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


# 最大子数组和求解
def maxSubArraySum(nums):
    dp = [0 for i in range(len(nums))]
    res = dp[0] = nums[0]

    for i in range(1, len(nums)):
        dp[i] = max(dp[i - 1], 0) + nums[i]
        res = max(res, dp[i])

    return res


# 将多行子矩阵，压缩为一维数组
def matrixZip(matrix):
    cols = len(matrix[0])
    rows = len(matrix)
    zip = [0 for i in range(cols)]

    for c in range(cols):
        for r in range(rows):
            zip[c] += matrix[r][c]

    return zip


# 算法入口
def getResult(n, m, matrix):
    dp = []

    for i in range(n):
        dp.append(maxSubArraySum(matrix[i]))
        for j in range(i + 1, n):
            dp.append(maxSubArraySum(matrixZip(matrix[i:j + 1])))

    dp.sort()

    return dp[-1]


# 算法调用
print(getResult(n, m, matrix))
```