一、二维数组

（一）考点分析

结构类型	重点考点	考察频率	难度
存储与遍历	行列下标范围计算	35%	★★
	内存连续性与缓存机制	25%	★★★
	边界越界风险	20%	★★★
子矩阵操作	子矩阵匹配算法	30%	★★★
	最大子矩阵和优化	25%	★★★★
	子矩阵位置映射	20%	★★
数学优化	二维前缀和应用	40%	★★★★
	差分矩阵扩展	15%	★★★★
	递推关系识别	10%	★★★
特殊矩阵	稀疏矩阵处理	10%	★★★
	蛇形矩阵填充	15%	★★
调试与优化	小数据模拟验证	30%	★★
	时间复杂度分析	25%	★★★
	空间预分配优化	10%	★★

（二）核心知识点

1. 存储与遍历

核心知识点：

- 行优先存储：内存中**按行连续存储**（a[0][0] → a[0][1] → a[1][0]）

[0][0]	**[0][1]**	**[0][2]**	**[0][3]**
[1][0]	**[1][1]**	**[1][2]**	**[1][3]**
[2][0]	**[2][1]**	**[2][2]**	**[2][3]**

按行存储

[0][0]	[0][1]	[0][2]	[0][3]	[1][0]	[1][1]	[1][2]	[1][3]	[2][0]	[2][1]	[2][2]	[2][3]

- 边界计算：子矩阵范围 [1, n1-n2+1]（行），[1, m1-m2+1]（列）
- 缓存优化：按行遍历比按列遍历**快 2-3 倍**

黄金法则：

```
// 正确遍历模板
for(int i=1; i<=n1-n2+1; i++)   // 行范围
  for(int j=1; j<=m1-m2+1; j++) // 列范围
```

i\j	1	2	3	4	5
1	1	2	3	4	5
2	6	7	8	9	0
3	1	2	3	4	5
4	6	7	8	9	0
5	1	2	3	4	5

i\j	1	2
1	8	9
2	3	4

高频公式：

```
// 子矩阵元素映射
a[i+k1-1][j+k2-1] == b[k1][k2]   // 2011 年 27 题核心
```

易错点：

- 漏掉 +1：写成 j<=m1-m2 导致漏掉最后一列
- 行列颠倒：a[j][i] 写成 a[i][j]
- 下标越界：未检查 i+k1 > n1

2. 子矩阵操作

核心知识点：

- 四重循环结构：外层定位（i,j）+ 内层验证（k1,k2）
- 匹配标记机制：good 变量初始化为 true,每个位置重新初始化
- 及时终止：发现不匹配立即 break

黄金法则：

```
bool good = true;   // 必须每个位置重新初始化
for(k1=1; k1<=n2; k1++){
  for(k2=1; k2<=m2; k2++){
    if(a[i+k1-1][j+k2-1] != b[k1][k2]){
      good = false;
      break;   // 及时退出提升效率
    }
  }
  if(!good) break; // 双 break 优化
}
```

1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4
4,1	4,2	4,3	4,4

高频公式：

```
// 子矩阵存在标记
haveAns = true;   // 找到至少一个匹配
```

易错点：

- 忘记重置 good：导致前次结果影响当前判断
- 冗余比较：未使用 break 继续无效比较
- 输出遗漏：忘记设置 haveAns 标记

3. 数学优化

核心知识点：

- 二维前缀和：**rowsum[i][j] = Σmatrix[i][1..j]**
- 降维打击：二维问题 → 一维最大子段和
- Kadane 算法：动态维护当前和

matrix	第1列	第2列
第1行	1	3
第2行	5	7
第3行	-2	-9

黄金法则：

```
3 2
1 3
5 7
-2 -9
16
```

rowsum	第0列	第1列	第2列
第1行	0	1	4
第2行	0	5	12
第3行	0	-2	-11

```
// 前缀和初始化
for(int i=1; i<=m; i++)
   rowsum[i][0] = 0;   // 关键！保证边界正确
// 前缀和构建
rowsum[i][j] = rowsum[i][j-1] + matrix[i][j];
```

高频公式：

```
3 2
1 3
5 7
-2 -9
16
```

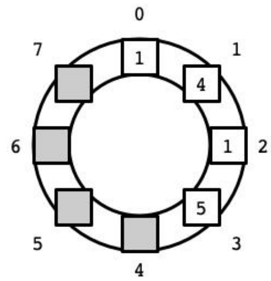
matrix	第1列	第2列
第1行	1	3
第2行	5	7
第3行	-2	-9

```
// 列区间和计算
area += rowsum[i][last] - rowsum[i][first-1]
// Kadane 算法核心
if(area > ans) ans = area;
if(area < 0) area = 0;   // 重置负区和
```

rowsum	第0列	第1列	第2列
第1行	0	1	4
第2行	0	5	12
第3行	0	-2	-11

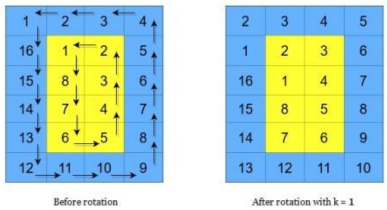
易错点：

- 前缀和索引错位：first-1 写成 first
- 初始化不全：漏掉 rowsum[i][0]=0
- 负值处理：未考虑全负矩阵，ans 初始值错误



4. 特殊矩阵处理

核心知识点：



- 环形矩阵：取模运算 (i+1)%n
- 蛇形矩阵：方向向量 dx={0,1,0,-1}, dy={1,0,-1,0}
- 稀疏矩阵：三元组存储 (row,col,value)

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

高频公式：

```
// 蛇形矩阵方向控制
int dx[4] = {0,1,0,-1}, dy[4] = {1,0,-1,0};
int dir = 0;
```

```
x += dx[dir]; y += dy[dir];
if(越界) dir = (dir+1)%4;   // 转向
```

易错点：

- 环形边界：忘记取模导致越界
- 方向切换：未检测已填充位置
- 稀疏矩阵：用二维数组存储浪费空间

5. 调试与优化

核心知识点：

- 小数据模拟：3x3 矩阵验证
- 复杂度分析：避免 $O(n^4)$ 算法
- 边界测试：n=0, n=1 特殊情况

黄金法则：

```
// 最大子矩阵和初始化
ans = matrix[1][1];   // 不能初始化为 0!
for(i=1; i<=m; i++)
  for(j=1; j<=n; j++)
    ans = max(ans, matrix[i][j]); // 防全负数
```

高频公式：

```
// 时间复杂度估算
子矩阵匹配：O(n1*m1*n2*m2) → 避免 n>50
最大子矩阵和：O(n²*m) → 可处理 n,m≤100
```

易错点：

- 未测试全负矩阵：ans 初始为 0 出错
- 越界访问：a[SIZE][SIZE] 但访问 [SIZE+1]

- 忽略空矩阵：未处理 n=0 的情况

核心程序模板

(1)子矩阵匹配模板

```
// 2011 年 27 题完整实现
bool haveAns = false;
for(i=1; i<=n1-n2+1; i++) {
  for(j=1; j<=m1-m2+1; j++) {
    bool good = true;   // 关键初始化
    for(k1=1; k1<=n2; k1++) {
      for(k2=1; k2<=m2; k2++) {
        if(a[i+k1-1][j+k2-1] != b[k1][k2]) {
          good = false;
          break;   // 优化点
        }
      }
      if(!good) break;   // 双 break 优化
    }
    if(good) {
      cout << i << " " << j << endl;
      haveAns = true;
    }
  }
}
if(!haveAns) cout << "There is no answer";
```

i\j	1	2	3	4	5
1	1	2	3	4	5
2	6	7	8	9	0
3	1	2	3	4	5
4	6	7	8	9	0
5	1	2	3	4	5

i\j	1	2
1	8	9
2	3	4

(2)最大子矩阵和模板

```
// 2014 年 28 题优化版
int ans = matrix[1][1];   // 防全负数
// 计算行前缀和
for(int i=1; i<=m; i++) {
    rowsum[i][0] = 0;   // 边界初始化
    for(int j=1; j<=n; j++) {
        rowsum[i][j] = rowsum[i][j-1] + matrix[i][j];
        ans = max(ans, matrix[i][j]);   // 单元素最大值
    }
}
// 枚举列区间
for(int L=1; L<=n; L++) {
    for(int R=L; R<=n; R++) {
        int area = 0;
        for(int i=1; i<=m; i++) {
            int rowVal = rowsum[i][R] - rowsum[i][L-1];
            area = max(rowVal, area + rowVal); // Kadane 算法
            ans = max(ans, area);
            if(area < 0) area = 0;   // 重置负区和
        }
    }
}
cout << ans;
```

```
3 2
1 3
5 7
-2 -9
16
```

matrix	第1列	第2列
第1行	1	3
第2行	5	7
第3行	-2	-9

rowsum	第0列	第1列	第2列
第1行	0	1	4
第2行	0	5	12
第3行	0	-2	-11

终极技巧：对于二维数组题，必做三件事

- 纸上画 3x3 矩阵模拟流程

- 检查四重循环边界（特别是 ±1）
- 测试全负/全零/单行单列极端情况

（三）真题强化

2011 年第 27 题（二维数组）

完善程序：

（子矩阵）给输入一个 n1m1 的矩阵 a，和 n2m2 的矩阵 b，问 a 中是否存在子矩阵和 b 相等。若存在，输出所有子矩阵左上角的坐标：若不存在输出 "There is no answer"。

```
5 5
1 2 3 4 5
6 7 8 9 0
1 2 3 4 5
6 7 8 9 0
1 2 3 4 5
2 2
8 9
3 4
2 3
4 3
```

i\j	1	2	3	4	5
1	1	2	3	4	5
2	6	7	8	9	0
3	1	2	3	4	5
4	6	7	8	9	0
5	1	2	3	4	5

i\j	1	2
1	8	9
2	3	4

```
#include<iostream>
using namespace std;
const int SIZE = 50;
int n1,m1,n2,m2,a[SIZE][SIZE],b[SIZE][SIZE];
int main()
{
    int i,j,k1,k2;
    bool good ,haveAns;
    cin>>n1>>m1;
    for(i=1;i<=n1;i++)
        for(j=1;j<=m1;j++)
            cin>>a[i][j];

    cin>>n2>>m2;
    for(i=1;i<=n2;i++)
        for(j=1;j<=m2;j++)
            [1]；  // ①
```

```
        haveAns=false;
        for(i=1;i<=n1-n2+1;i++)
            for(j=1;j<= [2] ;j++){   // ②
                [3] ;   // ③
                for(k1=1;k1<=n2;k1++)
                    for(k2=1;k2<= [4] ;k2++){   // ④
                        if(a[i+k1-1][j+k2-1]!=b[k1][k2])
                            good=false;
                    }
                if(good){
                    cout<<i<<' '<<j<<endl;
                    [5] ;   // ⑤
                }
            }
        if(!haveAns)
            cout<<"There is no answer"<<endl;
        return 0;
}
```

i\j	1	2	3	4	5
1	1	2	3	4	5
2	6	7	8	9	0
3	1	2	3	4	5
4	6	7	8	9	0
5	1	2	3	4	5

i\j	1	2
1	8	9
2	3	4

答案与解析

① 答案：cin >> b[i][j]

解析：此处需要读取矩阵 b 的元素。程序通过双重循环遍历矩阵 b 的每个位置，因此需要填入输入语句 cin >> b[i][j]。

② 答案：m1 - m2 + 1

解析：子矩阵的列遍历范围由大矩阵列数 m1 和小矩阵列数 m2 决定。最大起始列下标为 m1 - m2 + 1（如 m1=5, m2=2 时，j 范围是 1 到 4）。

③ 答案：good = true

解析：在检查每个子矩阵前，需初始化标记变量 good 为 true。若后续发现元素不匹配，则将其设为 false。

④ 答案：m2

解析：内层循环需遍历子矩阵的列（小矩阵 b 的列），因此循环上限为小矩阵列数 m2。

⑤ 答案：haveAns = true

解析：当找到匹配的子矩阵时，需将存在答案的标志 haveAns 设为 true，防止输出 "There is no answer"。

2014 年第 28 题（二维数组）

（最大子矩阵和）给出 m 行 n 列的整数矩阵，求最大的子矩阵和（子矩阵不能为空）。

输入第一行包含两个整数 m 和 n，即矩阵的行数和列数。之后 m 行，每行 n 个整数，描述整个矩阵。程序最终输出最大的子矩阵和。

（最后一空 4 分，其余 3 分，共 16 分）

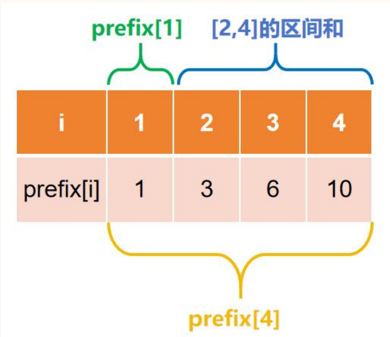
```
3 2
1 3
5 7
-2 -9
16
```

matrix	第1列	第2列
第1行	1	3
第2行	5	7
第3行	-2	-9

rowsum	第0列	第1列	第2列
第1行	0	1	4
第2行	0	5	12
第3行	0	-2	-11

```
#include <iostream>
using namespace std;
const int SIZE = 100;
int matrix[SIZE + 1][SIZE + 1];
int rowsum[SIZE + 1][SIZE + 1]; /* rowsum[i][j]记录第 i 行前 j 个数的和
*/
int m, n, i, j, first, last, area, ans;
int main()
{
 cin >> m >> n;
 for ( i = 1; i <= m; i++ )
     for ( j = 1; j <= n; j++ )
```

```
            cin >> matrix[i][j];
    ans = [1] ;   // ①
    for ( i = 1; i <= m; i++ )
        [2] ;   // ②
    for ( i = 1; i <= m; i++ )
        for ( j = 1; j <= n; j++ )
            rowsum[i][j] = [3] ;   // ③
    for ( first = 1; first <= n; first++ )
        for ( last = first; last <= n; last++ )
        {
            [4] ;   // ④
            for ( i = 1; i <= m; i++ )
            {
                area += [5] ;   // ⑤
                if ( area > ans )
                    ans = area;
                if ( area < 0 )
                    area = 0;
            }
        }
    cout << ans << endl;
    return(0);
}
```



答案与解析

① 答案：matrix[1][1]

解析：初始化最大和 ans 为矩阵第一个元素的值，确保后续比较时 ans 有初始值。

② 答案：rowsum[i][0] = 0

解析：初始化前缀和数组的第 0 列为 0，使得计算 rowsum[i][j] 时 j=1 的情况成立（如 rowsum[1][1] = rowsum[1][0] + matrix[1][1]）。

③ 答案：rowsum[i][j - 1] + matrix[i][j]

解析：计算第 i 行前 j 个元素的和，当前值 = 前 (j-1) 个元素的和 + 当前元素值。

④ 答案：area = 0

解析：在计算新的列范围 (first 到 last) 的子矩阵和之前，需将临时累加变量 area 归零。

⑤ 答案：rowsum[i][last] - rowsum[i][first - 1]

解析：计算第 i 行从 first 列到 last 列的和，利用前缀和数组可直接用 rowsum[i][last] - rowsum[i][first - 1] 高效求得(如 first=2, last=3 时：rowsum[i][3] - rowsum[i][1])。

完整版程序

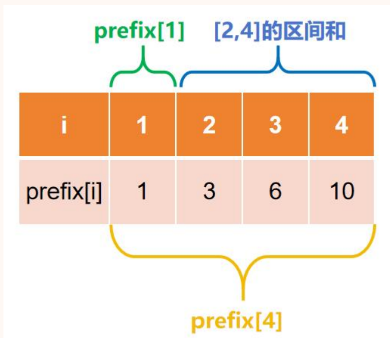
```
#include <iostream>
using namespace std;
const int SIZE = 100;
int matrix[SIZE + 1][SIZE + 1];
int rowsum[SIZE + 1][SIZE + 1]; /* rowsum[i][j]记录第 i 行前 j 个数的和
*/
int m, n, i, j, first, last, area, ans;
int main()
{
 cin >> m >> n;
 for ( i = 1; i <= m; i++ )
     for ( j = 1; j <= n; j++ )
```

```
3 2
1 3
5 7
-2 -9
16
```

matrix	第1列	第2列
第1行	1	3
第2行	5	7
第3行	-2	-9

rowsum	第0列	第1列	第2列
第1行	0	1	4
第2行	0	5	12
第3行	0	-2	-11

```
            cin >> matrix[i][j];
    ans = matrix[1][1] ;   // ①
    for ( i = 1; i <= m; i++ )
        rowsum[i][0] = 0 ;   // ②
    for ( i = 1; i <= m; i++ )
        for ( j = 1; j <= n; j++ )
            rowsum[i][j] = rowsum[i][j - 1] + matrix[i][j] ;   // ③
    for ( first = 1; first <= n; first++ )
        for ( last = first; last <= n; last++ )  {
        area = 0;   // ④
        for ( i = 1; i <= m; i++ ){
            area += rowsum[i][last] - rowsum[i][first - 1] ;   // ⑤
            if ( area > ans )
                ans = area;
            if ( area < 0 )
                area = 0;
        }
    }
    cout << ans << endl;
    return(0);
}
```



程序的目标是计算该矩阵的最大子矩阵和（即所有可能的连续子矩阵中，元素和的最大值）。程序使用了一种优化方法：通过计算每行的前缀和（prefix sum），然后枚举所有可能的列范围（从列 first 到列 last），并在这些列范围内应用 Kadane 算法（一种高效计算最大子数组和的算法）来求取最大子矩阵和。

程序执行步骤

假设输入为 m = 3（行数）和 n = 2（列数），矩阵元素按行输入：1 3 5 7 -2 -9。以下是逐步执行过程：

1. 输入读取和初始化

- cin >> m >> n; 读取行数 m = 3 和列数 n = 2。
- 嵌套循环读取矩阵元素到 matrix[i][j] 数组中（索引从 1 开始）：
  - matrix[1][1] = 1, matrix[1][2] = 3
  - matrix[2][1] = 5, matrix[2][2] = 7
  - matrix[3][1] = -2, matrix[3][2] = -9
- ans = matrix[1][1]; 初始化 ans（存储最大子矩阵和）为 matrix[1][1] = 1。这是为了确保 ans 至少包含矩阵中的一个元素（作为初始候选最大值）。

2. 计算行前缀和（rowsum 数组）

- rowsum[i][j] 表示第 i 行前 j 个元素的和（即前缀和）。这用于快速计算任意行在列范围 [first, last] 的和。
- 初始化：rowsum[i][0] = 0（每行的前缀和基准，索引 0 为 0）。
- 计算前缀和（嵌套循环）：
  - 公式：rowsum[i][j] = rowsum[i][j-1] + matrix[i][j]
- 针对当前矩阵的计算结果：
  - 行 1（i=1）：
    - j=1: rowsum[1][1] = rowsum[1][0] + matrix[1][1] = 0 + 1 = 1
    - j=2: rowsum[1][2] = rowsum[1][1] + matrix[1][2] = 1 + 3 = 4
  - 行 2（i=2）：
    - j=1: rowsum[2][1] = 0 + 5 = 5
    - j=2: rowsum[2][2] = 5 + 7 = 12
  - 行 3（i=3）：
    - j=1: rowsum[3][1] = 0 + (-2) = -2
    - j=2: rowsum[3][2] = -2 + (-9) = -11
- 最终 rowsum 数组为：

rowsum	第0列	第1列	第2列
第1行	0	1	4
第2行	0	5	12
第3行	0	-2	-11

3. 枚举列范围并计算最大子矩阵和

程序通过两个外层循环枚举所有可能的列范围 [first, last]（first 是起始列，last 是结束列）。对于每个列范围，它计算一个“压缩数组”，其中每个元素是第 i 行在列 [first, last] 的和（通过 rowsum[i][last] - rowsum[i][first-1] 快速得到）。然后，对这个压缩数组应用 Kadane 算法（在行方向上遍历），计算最大子数组和，并更新 ans。

由于 n=2，可能的列范围有三种情况：

Case 1: first = 1, last = 1（只包含第 1 列）

- 压缩数组：每行在第 1 列的和，即 [matrix[1][1], matrix[2][1], matrix[3][1]] = [1, 5, -2]。
- 计算过程（初始化 area = 0，用于 Kadane 算法）：
  - i=1: area += (rowsum[1][1] - rowsum[1][0]) = 1 - 0 = 1 → area = 1
    - area (1) > ans (1)? 否（相等），ans 不变（仍为 1）。
    - area (1) >= 0，不重置。
  - i=2: area += (rowsum[2][1] - rowsum[2][0]) = 5 - 0 = 5 → area = 1 + 5 = 6
    - area (6) > ans (1)? 是，ans = 6。
    - area (6) >= 0，不重置。
  - i=3: area += (rowsum[3][1] - rowsum[3][0]) = -2 - 0 = -2 → area = 6 + (-2) = 4
    - area (4) > ans (6)? 否。
    - area (4) >= 0，不重置。
- 此列范围的最大子数组和（对应子矩阵：行 1-2、列 1）为 6。

Case 2: first = 1, last = 2（包含所有列）

- 压缩数组：每行所有列的和，即 [1+3, 5+7, (-2)+(-9)] = [4, 12, -11]。

- 计算过程（初始化 area = 0）：
  - i=1: area += (rowsum[1][2] - rowsum[1][0]) = 4 - 0 = 4 → area = 4
    - area (4) > ans (6)? 否，ans 不变（仍为 6）。
    - area (4) >= 0，不重置。
  - i=2: area += (rowsum[2][2] - rowsum[2][0]) = 12 - 0 = 12 → area = 4 + 12 = 16
    - area (16) > ans (6)? 是，ans = 16。
    - area (16) >= 0，不重置。
  - i=3: area += (rowsum[3][2] - rowsum[3][0]) = -11 - 0 = -11 → area = 16 + (-11) = 5
    - area (5) > ans (16)? 否。
    - area (5) >= 0，不重置。
- 此列范围的最大子数组和（对应子矩阵：行 1-2、所有列）为 16。

Case 3: first = 2, last = 2（只包含第 2 列）

- 压缩数组：每行在第 2 列的和，即 [matrix[1][2], matrix[2][2], matrix[3][2]] = [3, 7, -9]。
- 计算过程（初始化 area = 0）：
  - i=1: area += (rowsum[1][2] - rowsum[1][1]) = 4 - 1 = 3 → area = 3
    - area (3) > ans (16)? 否。
    - area (3) >= 0，不重置。
  - i=2: area += (rowsum[2][2] - rowsum[2][1]) = 12 - 5 = 7 → area = 3 + 7 = 10
    - area (10) > ans (16)? 否。
    - area (10) >= 0，不重置。
  - i=3: area += (rowsum[3][2] - rowsum[3][1]) = -11 - (-2) = -9 → area = 10 + (-9) = 1
    - area (1) > ans (16)? 否。

  - area (1) >= 0，不重置。
- 此列范围的最大子数组和（对应子矩阵：行 1-2、列 2）为 10。

4. 输出结果

- 在所有列范围处理完毕后，ans 存储了最大子矩阵和。
- cout << ans << endl; 输出 ans = 16。