

1. 单条指令用时 $t_0 = 10^{-8}s$
2. 时间限制 $t_{limit} = 2t_{max}$, 则 $t_{max} = \frac{t_{limit}}{2}$
3. 设 $c(n)$ 为输入规模为 n 的时候执行的指令条数, 那么有: $t_{max} = c(n_{max})t_0$, 则 $t_{limit} = 2c(n_{max})t_0$

2A

$t_{limit} = 1s$, 则 $2 \times 10n^2 \times 10^{-8} = 1$, 则 $n^2 = 5 \times 10^6$

$n = \sqrt{5 \times 10^6} \approx 2236.06$, 由于条件要求不超过, 故设置的最大输入规模应该为2236

2B

$c(n) = 20n \log_2 n$, 则有 $2 \times 20n \log_2 n \times 10^{-8} = 1$, 则 $n \log_2 n = 2.5 \times 10^6$

$n \approx 145746.2$, 则 n 应该取145746

3A

solution_1.cpp:

- bug1: sum变量没有在每次查询之前进行重置, 导致后面的运行结果会收到前面的结果的干扰
发现方式:

1. 静态检查: 观察代码并且复述解题逻辑, 容易发现每次循环之前应该重置sum
2. 观察输出结果: 输入测试样例, 发现第一个输出结果正确, 之后的输出结果显著大于答案, 容易想到应该是由于每次查询的初始化导致错误

- bug2: 数据类型选择错误

根据题目给出的数据范围, sum值最大可能到达 $4e11$, 而程序中使用了int型的sum, 最大只能储存约 $2e9$ 大小的数据, 有很大概率会导致溢出

发现方式:

1. 静态检查
2. ai辅助

solution_2.cpp:

- bug1: sum 变量未在每次查询前重置 (同 solution_1)
- bug2: 数据类型选择错误 (同solution_1)
- bug3: 前缀和数组定义为 `rowsum[i][j]` 代表第*i*行的[0, j]列的和, 而代码中使用了 `rowsum[x+j][y+b] - rowsum[x+j][y]` 这样就没有包含第*y*列, 但是多包含了第*y+b*列, 导致出错

发现方式:

- 静态检查: 前缀和公式与子矩阵范围不匹配。
- 输出调试: 打印每次累加的行和, 发现与手动计算的子矩阵列范围不符。
- 对拍: 之前已经调试得到了正确的solution_1, 通过solution_1正确输出来进行比较

AIGC 工具调试能力

- 优点：
 - 可以快速识别一些细节错误，比如语法错误、循环变量初始化、索引错误等
 - 对于一些经典的算法调试能力很强，比如前缀和
 - 对于一些常见的错误调试能力强，比如索引错误，循环变量，数据溢出等
 - 对于一些符合编程范式，比较规范的代码可以快速匹配已有的数据来发现错误
- 局限性
 - 对于一些比较复杂的逻辑调试能力较弱，容易出现逻辑混乱
 - 对于具有较长上下文的代码理解能力比较弱，难以结合具体场景进行调试
 - 对于极端情况，一些边界条件容易忽略
 - 对于一些非典型的错误，比如一些经过多次修改和补丁的代码调试能力较弱
 - 对于一些运行时错误调试能力较弱

3B

在编译时加上-g选项来保存调试信息，如 `g++ solution_1.cpp -o solution_1 -g`

1. 启动调试： `gdb ./solution_1`
2. 打断点： `break <行号>/<函数名>`，比如使用 `break main` 在 `main` 函数入口处打断点，使用 `break 15` 在第15行打断点
3. 运行程序：使用 `run` 开始运行
4. 单步执行： `next` 是逐过程， `step` 是逐语句
5. 查看变量： `print sum` 打印 `sum` 值， `watch sum` 监控 `sum` 值变化
6. `continue`：运行到下一个断点

3C

`srand(time(0))`

`srand(seed)` 用于设定产生随机数的种子，不同的种子对应不同的随机数序列

`time(0)` 则是用于获取当前的系统时间

两者结合起来可以实现在不同的时间运行程序都会给随机数生成器不同的种子，从而实现每次运行程序都会产生不同的测试数据

3D

1. `system("g++ rand_input.cpp -o rand_input");`
对 `rand_input.cpp` 进行编译，输出结果为 `rand_input`，这里的 `rand_input` 是用于随机生成测试数据的程序
2. `system("g++ check_input.cpp -o check_input");`
对 `check_input.cpp` 进行编译，输出结果为 `check_input`，这里的 `check_input` 是用于检查生成的数据是否合法的程序
3. `system("g++ solution_1.cpp -o solution_1");`
对 `solution_1.cpp` 进行编译，输出结果为 `solution_1_input`

4. `system("g++ solution_2.cpp -o solution_2");`

对solution_2.cpp进行编译，输出结果为solution_2

5. `system("./rand_input > rand.in");`

执行rand_input，生成随机测试数据，并且将输出内容重定向到rand.in中

6. `system("./check_input < rand.in")!=0`

将rand.in中的数据输入check_in中进行验证，check_input中使用了assert来判断，如果输入内容不合法则会异常退出，main函数的返回值是非零值，结束循环，否则return 0 继续

7. `system("./solution_1 < rand.in > 1.out");`

`system("./solution_2 < rand.in > 2.out");`

执行solution_1和2，将rand.in作为输入数据，将输出内容重定向输出到1.out和2.out中

8. `system("diff 1.out 2.out")!=0`

比较两个程序的输出是否一致，若不一致则终止循环并提示

3E

- 最大可能值
 - 单个元素最大绝对值： $1e5$
 - 当全选矩阵的时候取到子矩阵最大规模： $2000 \times 2000 = 4e6$
 - 最大查询次数： $1e5$
 - 总和上限： $1e5 \times 4e6 = 4e11$
 - 如果按照给出的solution中错误的代码，每次没有重置sum，那么总和上限则是： $1e5 \times 4e6 \times 1e5 = 4e16$
- 数据类型：int类型为32位，最大值为2147483647，约为 $2e9$ ，远远小于总和上限，所以需使用64位整数long long存储

4A

思路：使用二维前缀和数组

1. 预处理，建立前缀和数组

定义二维数组 `prefixsum[i][j]` 表示从矩阵左上角 `(1,1)` 到 `(i,j)` 的矩形区域内所有元素的和。

构建公式 `prefixsum[i][j]=matrix[i][j]+prefixsum[i-1][j]+prefixsum[i][j-1]-prefixsum[i-1][j-1]`

(这里0行0列都存0，方便边界处理)

2. 查询计算：

对于起点为 `(x,y)`、大小为 `(a,b)` 的子矩阵（终点为 `x+a-1, y+b-1`），其和为：

`sum=prefixsum[x+a-1][y+b-1]-prefixsum[x-1][y+b-1]-prefixsum[x+a-1][y-1]+prefixsum[x-1][y-1]`，时间复杂度为常数级别

4B

1. test1

原数组大，查询次数多，子数组大

n	m	p	a	b
10000	10000	1000000	8000	8000

测试结果：

	1	2	3	平均
solution_1	1653ms	1534ms	1609ms	1598ms
solution_2	1945ms	1897ms	1808ms	1883ms
solution_3	584ms	545ms	551ms	560ms

2. test2

原数组大，查询次数少，子数组小

n	m	p	a	b
2000	20000	1	10	10

测试结果：

	1	2	3	平均
solution_1	828ms	840ms	812ms	827ms
solution_2	870ms	875ms	861ms	869ms
solution_3	1034ms	997ms	972ms	1001ms

3. test3

原数组小，查询次数多，子数组行大列小

n	m	p	a	b
2000	20	1000000	1900	10

测试结果：

	1	2	3	平均
solution_1	27271ms	27186ms	26002ms	26820ms
solution_2	3527ms	3483ms	3380ms	3463ms

	1	2	3	平均
solution_3	101ms	98ms	101ms	100ms

4c

程序	预处理时间复杂度	单次查询时间复杂度	总时间复杂度 (n, m 为矩阵规模, a, b 为子矩阵规模, p 为查询次数)
solution_1	$O(1)$	$O(ab)$	$O(abp)$
solution_2	$O(nm)$	$O(a)$	$O(nm+ap)$
solution_3	$O(nm)$	$O(1)$	$O(nm+p)$

solution_1在预处理的时间复杂度低, 2, 3在预处理的时间复杂度较高

solution_1查询的时间复杂度较高, 2相对较低, 但是对子数组列数敏感, 3查询可在常数时间内完成

solution_3优势条件:

1. 原数组大小: 当原数组大小较小时, 预处理所用时间较少, 后续查询成本可以摊薄预处理成本
2. 查询次数: 当查询次数极大时, 由于3的查询效率高于1和2, 所以当查询次数多的时候solution_3有明显优势 (当子数组的行数较少时, 2, 3的差距较小)
3. 子数组大小: 当子数组规模大时, 3有优势。子数组规模对3的查询没有影响, 对于1, 子数组规模对查询成本影响较大, 对于2, 子数组的行数影响较大

5A

网页代码:

```

1 <!DOCTYPE html>
2 <html lang="zh-CN">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>二维前缀和算法可视化 (solution_3.cpp)</title>
7   <script src="https://cdn.tailwindcss.com"></script>
8   <style>
9     /* 矩阵单元格样式 */
10    .matrix-cell {
11      width: 40px;
12      height: 40px;
13      text-align: center;
14      border: 1px solid #333;
15      transition: all 0.3s ease;
16    }
17    /* 前缀和矩阵高亮 (计算中) */
18    .prefix-highlight {

```

```

19         background-color: #93c5fd;
20         font-weight: bold;
21     }
22     /* 查询子矩阵高亮 */
23     .query-highlight {
24         background-color: #fbbf24;
25     }
26     /* 步骤指示器样式 */
27     .step-active {
28         background-color: #2563eb;
29         color: white;
30         font-weight: bold;
31     }
32 </style>
33 </head>
34 <body class="bg-gray-50 p-5">
35     <div class="max-w-7xl mx-auto">
36         <h1 class="text-3xl font-bold text-center text-gray-800 mb-8">二维前
缀和算法可视化 (solution_3.cpp)</h1>
37
38         <!-- 控制区：参数配置与操作按钮 -->
39         <div class="bg-white rounded-lg shadow-md p-5 mb-6">
40             <h2 class="text-xl font-semibold text-gray-700 mb-4">1. 配置参数
</h2>
41             <div class="grid grid-cols-1 md:grid-cols-4 gap-4 mb-4">
42                 <div>
43                     <label class="block text-sm font-medium text-gray-600
mb-1">矩阵行数 n</label>
44                     <input type="number" id="n" value="4" min="2" max="10"
class="w-full p-2 border border-gray-300 rounded">
45                 </div>
46                 <div>
47                     <label class="block text-sm font-medium text-gray-600
mb-1">矩阵列数 m</label>
48                     <input type="number" id="m" value="4" min="2" max="10"
class="w-full p-2 border border-gray-300 rounded">
49                 </div>
50                 <div>
51                     <label class="block text-sm font-medium text-gray-600
mb-1">查询次数 q</label>
52                     <input type="number" id="q" value="2" min="1" max="5"
class="w-full p-2 border border-gray-300 rounded">
53                 </div>
54                 <div class="flex items-end">
55                     <button id="generate-btn" class="w-full bg-blue-500
text-white py-2 px-4 rounded hover:bg-blue-600 transition">生成矩阵</button>
56                 </div>
57             </div>
58
59             <h2 class="text-xl font-semibold text-gray-700 mb-4">2. 算法步骤
控制</h2>
60             <div class="flex flex-wrap gap-3 mb-4">
61                 <button id="preprocess-btn" class="bg-green-500 text-white
py-2 px-4 rounded hover:bg-green-600 transition disabled">1. 计算二维前缀和
</button>

```

```

62         <button id="next-query-btn" class="bg-purple-500 text-white
py-2 px-4 rounded hover:bg-purple-600 transition" disabled>2. 执行下一次查询
</button>
63         <button id="reset-btn" class="bg-red-500 text-white py-2
px-4 rounded hover:bg-red-600 transition">重置</button>
64     </div>
65
66     <!-- 查询参数配置（动态显示） -->
67     <div id="query-config" class="hidden bg-gray-50 p-4 rounded mb-
4">
68         <h3 class="text-lg font-medium text-gray-700 mb-2">当前查询参
数（1-based）</h3>
69         <div class="grid grid-cols-1 md:grid-cols-4 gap-4">
70             <div>
71                 <label class="block text-sm font-medium text-gray-
600 mb-1">起始行 x</label>
72                 <input type="number" id="query-x" min="1" class="w-
full p-2 border border-gray-300 rounded">
73             </div>
74             <div>
75                 <label class="block text-sm font-medium text-gray-
600 mb-1">起始列 y</label>
76                 <input type="number" id="query-y" min="1" class="w-
full p-2 border border-gray-300 rounded">
77             </div>
78             <div>
79                 <label class="block text-sm font-medium text-gray-
600 mb-1">子矩阵高度 a</label>
80                 <input type="number" id="query-a" min="1" class="w-
full p-2 border border-gray-300 rounded">
81             </div>
82             <div>
83                 <label class="block text-sm font-medium text-gray-
600 mb-1">子矩阵宽度 b</label>
84                 <input type="number" id="query-b" min="1" class="w-
full p-2 border border-gray-300 rounded">
85             </div>
86         </div>
87         <button id="exec-query-btn" class="mt-3 bg-purple-500 text-
white py-2 px-4 rounded hover:bg-purple-600 transition">执行查询</button>
88     </div>
89 </div>
90
91 <!-- 可视化区：原始矩阵与前缀和矩阵 -->
92 <div class="grid grid-cols-1 lg:grid-cols-2 gap-6 mb-6">
93     <!-- 原始矩阵 -->
94     <div class="bg-white rounded-lg shadow-md p-5">
95         <h2 class="text-xl font-semibold text-gray-700 mb-4">原始矩阵
(v(i,j))</h2>
96         <div id="original-matrix" class="flex flex-col items-center
gap-1"></div>
97     </div>
98     <!-- 前缀和矩阵 -->
99     <div class="bg-white rounded-lg shadow-md p-5">
100         <h2 class="text-xl font-semibold text-gray-700 mb-4">二维前缀
和矩阵（prefixsum[i][j]）</h2>

```

```

101         <div class="text-sm text-gray-500 mb-2">公式:  $prefixsum[i][j] = v[i][j] + prefixsum[i-1][j] + prefixsum[i][j-1] - prefixsum[i-1][j-1]$ 
102     </div>
103     <div id="prefix-matrix" class="flex flex-col items-center gap-1"></div>
104 </div>
105
106 <!-- 结果区: 查询记录与算法说明 -->
107 <div class="bg-white rounded-lg shadow-md p-5">
108     <h2 class="text-xl font-semibold text-gray-700 mb-4">3. 查询结果
109 记录</h2>
110
111     <div id="query-results" class="space-y-3 mb-6"></div>
112
113     <h2 class="text-xl font-semibold text-gray-700 mb-4">4. 算法说明
114 (对应 solution_3.cpp) </h2>
115     <div class="text-gray-700 space-y-2">
116         <p><span class="font-bold">预处理阶段</span>: 时间复杂度  $O(nxm)$ , 构建二维前缀和矩阵, 将子矩阵求和的计算成本转移到初始化阶段。</p>
117         <p><span class="font-bold">查询阶段</span>: 时间复杂度  $O(1)$ , 通过前缀和公式计算子矩阵和: </p>
118         <p class="p1-5">sum = prefixsum[end_x][end_y] - prefixsum[x-1][end_y] - prefixsum[end_x][y-1] + prefixsum[x-1][y-1]</p>
119         <p>其中, end_x = x + a - 1 (子矩阵右下角行), end_y = y + b - 1 (子矩阵右下角列)。</p>
120     </div>
121 </div>
122 <script>
123     // 全局变量: 存储矩阵数据、查询状态
124     let originalMatrix = []; // 原始矩阵
125     let prefixMatrix = []; // 前缀和矩阵
126     let currentQuery = 0; // 当前执行的查询序号 (从1开始)
127     let totalQueries = 0; // 总查询次数
128
129     // DOM 元素获取
130     const nInput = document.getElementById('n');
131     const mInput = document.getElementById('m');
132     const qInput = document.getElementById('q');
133     const generateBtn = document.getElementById('generate-btn');
134     const preprocessBtn = document.getElementById('preprocess-btn');
135     const nextQueryBtn = document.getElementById('next-query-btn');
136     const resetBtn = document.getElementById('reset-btn');
137     const queryConfig = document.getElementById('query-config');
138     const queryX = document.getElementById('query-x');
139     const queryY = document.getElementById('query-y');
140     const queryA = document.getElementById('query-a');
141     const queryB = document.getElementById('query-b');
142     const execQueryBtn = document.getElementById('exec-query-btn');
143     const originalMatrixEl = document.getElementById('original-matrix');
144     const prefixMatrixEl = document.getElementById('prefix-matrix');
145     const queryResultsEl = document.getElementById('query-results');
146
147     // 1. 生成随机原始矩阵

```



```

147 generateBtn.addEventListener('click', () => {
148     const n = parseInt(nInput.value);
149     const m = parseInt(mInput.value);
150     totalQueries = parseInt(qInput.value);
151
152     // 重置状态
153     originalMatrix = [];
154     prefixMatrix = [];
155     currentQuery = 0;
156     originalMatrixEl.innerHTML = '';
157     prefixMatrixEl.innerHTML = '';
158     queryResultsEl.innerHTML = '';
159     queryConfig.classList.add('hidden');
160
161     // 生成随机矩阵（值范围 0~20，便于可视化）
162     for (let i = 1; i <= n; i++) {
163         const row = [];
164         for (let j = 1; j <= m; j++) {
165             row.push(Math.floor(Math.random() * 21)); // 0~20
166         }
167         originalMatrix.push(row);
168     }
169
170     // 渲染原始矩阵（1-based 显示）
171     for (let i = 0; i < originalMatrix.length; i++) {
172         const rowEl = document.createElement('div');
173         rowEl.className = 'flex gap-1';
174         for (let j = 0; j < originalMatrix[i].length; j++) {
175             const cellEl = document.createElement('div');
176             cellEl.className = 'matrix-cell flex items-center
justify-center';
177             cellEl.textContent = originalMatrix[i][j];
178             cellEl.id = `orig-${i+1}-${j+1}`; // 1-based 编号
179             rowEl.appendChild(cellEl);
180         }
181         originalMatrixEl.appendChild(rowEl);
182     }
183
184     // 启用预处理按钮
185     preprocessBtn.disabled = false;
186     nextQueryBtn.disabled = true;
187 });
188
189 // 2. 计算并渲染二维前缀和矩阵（带步骤高亮）
190 preprocessBtn.addEventListener('click', async () => {
191     const n = originalMatrix.length;
192     const m = originalMatrix[0].length;
193
194     // 初始化前缀和矩阵（0行0列为0，便于计算）
195     prefixMatrix = Array(n + 1).fill(0).map(() => Array(m +
1).fill(0));
196
197     // 清空前缀和矩阵容器
198     prefixMatrixEl.innerHTML = '';
199
200     // 渲染 0 行（辅助行，灰色显示）

```

```

201     const zeroRow = document.createElement('div');
202     zeroRow.className = 'flex gap-1';
203     zeroRow.appendChild(document.createElement('div')); // 空单元格
    (对应0行0列)
204     for (let j = 1; j <= m; j++) {
205         const cellE1 = document.createElement('div');
206         cellE1.className = 'matrix-cell flex items-center justify-
center text-gray-400';
207         cellE1.textContent = '0';
208         zeroRow.appendChild(cellE1);
209     }
210     prefixMatrixEl.appendChild(zeroRow);
211
212     // 逐行逐列计算前缀和 (1-based)，并高亮显示计算过程
213     for (let i = 1; i <= n; i++) {
214         const rowE1 = document.createElement('div');
215         rowE1.className = 'flex gap-1';
216
217         // 渲染 0 列 (辅助列, 灰色显示)
218         const zeroCell = document.createElement('div');
219         zeroCell.className = 'matrix-cell flex items-center
justify-center text-gray-400';
220         zeroCell.textContent = '0';
221         rowE1.appendChild(zeroCell);
222
223         // 渲染当前行的前缀和单元格
224         for (let j = 1; j <= m; j++) {
225             const cellE1 = document.createElement('div');
226             cellE1.className = 'matrix-cell flex items-center
justify-center';
227             cellE1.id = `prefix-${i}-${j}`;
228             rowE1.appendChild(cellE1);
229         }
230         prefixMatrixEl.appendChild(rowE1);
231
232         // 逐列计算并更新 (添加延迟, 便于观察)
233         for (let j = 1; j <= m; j++) {
234             const v = originalMatrix[i-1][j-1]; // 原始矩阵是0-based
    数组
235             const up = prefixMatrix[i-1][j];
236             const left = prefixMatrix[i][j-1];
237             const upLeft = prefixMatrix[i-1][j-1];
238             prefixMatrix[i][j] = v + up + left - upLeft;
239
240             // 高亮当前计算的单元格
241             const currentCell =
document.getElementById(`prefix-${i}-${j}`);
242             currentCell.classList.add('prefix-highlight');
243             await delay(500); // 延迟500ms, 观察计算步骤
244
245             // 更新单元格值, 并移除高亮
246             currentCell.textContent = prefixMatrix[i][j];
247             currentCell.classList.remove('prefix-highlight');
248         }
249     }
250

```

```

251 // 预处理完成，启用查询按钮
252 preprocessBtn.disabled = true;
253 nextQueryBtn.disabled = false;
254 });
255
256 // 3. 执行下一次查询（显示配置面板）
257 nextQueryBtn.addEventListener('click', () => {
258     currentQuery++;
259     if (currentQuery > totalQueries) {
260         alert('已完成所有查询!');
261         nextQueryBtn.disabled = true;
262         return;
263     }
264
265     // 显示查询配置面板，并设置参数范围（防止越界）
266     queryConfig.classList.remove('hidden');
267     const n = originalMatrix.length;
268     const m = originalMatrix[0].length;
269     queryX.max = n;
270     queryY.max = m;
271     queryX.value = 1; // 默认起始行1
272     queryY.value = 1; // 默认起始列1
273     queryA.max = n - queryX.value + 1;
274     queryB.max = m - queryY.value + 1;
275     queryA.value = 2; // 默认高度2
276     queryB.value = 2; // 默认宽度2
277
278     // 监听起始坐标变化，更新子矩阵大小的最大值
279     queryX.addEventListener('input', updateQueryMax);
280     queryY.addEventListener('input', updateQueryMax);
281 });
282
283 // 更新查询参数的最大值（防止子矩阵越界）
284 function updateQueryMax() {
285     const n = originalMatrix.length;
286     const m = originalMatrix[0].length;
287     const x = parseInt(queryX.value);
288     const y = parseInt(queryY.value);
289     queryA.max = n - x + 1;
290     queryB.max = m - y + 1;
291 }
292
293 // 4. 执行查询（计算子矩阵和，并高亮显示）
294 execQueryBtn.addEventListener('click', () => {
295     const x = parseInt(queryX.value);
296     const y = parseInt(queryY.value);
297     const a = parseInt(queryA.value);
298     const b = parseInt(queryB.value);
299     const endX = x + a - 1;
300     const endY = y + b - 1;
301     const n = originalMatrix.length;
302     const m = originalMatrix[0].length;
303
304     // 校验参数合法性
305     if (endX > n || endY > m) {
306         alert('查询参数越界！子矩阵超出原始矩阵范围');

```

```

307         return;
308     }
309
310     // 计算子矩阵和 (O(1) 公式)
311     const sum = prefixMatrix[endX][endY]
312         - prefixMatrix[x-1][endY]
313         - prefixMatrix[endX][y-1]
314         + prefixMatrix[x-1][y-1];
315
316     // 高亮原始矩阵中的查询子矩阵
317     for (let i = x; i <= endX; i++) {
318         for (let j = y; j <= endY; j++) {
319             const cell = document.getElementById(`orig-${i}-${j}`);
320             cell.classList.add('query-highlight');
321         }
322     }
323
324     // 高亮前缀和矩阵中的关键位置 (公式用到的4个点)
325     const highlightPoints = [
326         [endX, endY],
327         [x-1, endY],
328         [endX, y-1],
329         [x-1, y-1]
330     ];
331     highlightPoints.forEach(([i, j]) => {
332         if (i >= 0 && j >= 0) { // 0行0列也需要高亮
333             const cell =
334 document.getElementById(`prefix-${i}-${j}`) ||
335 Array.from(prefixMatrixEl.children[i].children)[j+1];
336             if (cell) cell.classList.add('query-highlight');
337         }
338     });
339
340     // 添加查询结果到记录
341     const resultItem = document.createElement('div');
342     resultItem.className = 'p-3 border border-gray-200 rounded';
343     resultItem.innerHTML = `
344         <div class="font-medium">查询
345     ${currentQuery}/${totalQueries}</div>
346         <div>参数: 起始坐标 (${x}, ${y}), 子矩阵大小 (${a}, ${b}), 右下角
347     坐标 (${endX}, ${endY})</div>
348         <div>子矩阵和: ${sum}</div>
349         <div class="text-sm text-gray-500">计算过程:
350     prefixsum[${endX}][${endY}] - prefixsum[${x-1}][${endY}] -
351     prefixsum[${endX}][${y-1}] + prefixsum[${x-1}][${y-1}] =
352     ${prefixMatrix[endX][endY]} - ${prefixMatrix[x-1][endY]} -
353     ${prefixMatrix[endX][y-1]} + ${prefixMatrix[x-1][y-1]} = ${sum}</div>
354     `;
355     queryResultsEl.appendChild(resultItem);
356
357     // 隐藏查询配置面板, 更新按钮状态
358     queryConfig.classList.add('hidden');
359     if (currentQuery >= totalQueries) {
360         nextQueryBtn.disabled = true;
361     }
362 }
363 });

```

```
355
356 // 5. 重置所有状态
357 resetBtn.addEventListener('click', () => {
358     location.reload(); // 简单重置：刷新页面
359 });
360
361 // 工具函数：延迟函数（用于步骤可视化）
362 function delay(ms) {
363     return new Promise(resolve => setTimeout(resolve, ms));
364 }
365 </script>
366 </body>
367 </html>
```

截图：

1. 生成矩阵：

二维前缀和算法可视化 (solution_3.cpp)

1. 配置参数

矩阵行数 n4 矩阵列数 m4 查询次数 q2 生成矩阵

2. 算法步骤控制

1. 计算二维前缀和 2. 执行下一次查询 重置

原始矩阵 (v(i,j))

19	7	17	4
10	9	10	14
8	2	14	14
13	2	10	8

二维前缀和矩阵 (prefixsum[i][j])

公式: $prefixsum[i][j] = v[i][j] + prefixsum[i-1][j] + prefixsum[i][j-1] - prefixsum[i-1][j-1]$

3. 查询结果记录

4. 算法说明 (对应 solution_3.cpp)

预处理阶段: 时间复杂度 $O(n \times m)$, 构建二维前缀和矩阵, 将子矩阵求和的计算成本转移到初始化阶段。

查询阶段: 时间复杂度 $O(1)$, 通过前缀和公式计算子矩阵和:

2. 计算前缀和

二维前缀和算法可视化 (solution_3.cpp)

1. 配置参数

矩阵行数 n4 矩阵列数 m4 查询次数 q2 生成矩阵

2. 算法步骤控制

1. 计算二维前缀和 2. 执行下一次查询 重置

原始矩阵 (v(i,j))

2	15	15	2
20	5	12	18
5	0	20	7
8	5	18	1

二维前缀和矩阵 (prefixsum[i][j])

公式: $prefixsum[i][j] = v[i][j] + prefixsum[i-1][j] + prefixsum[i][j-1] - prefixsum[i-1][j-1]$

0	0	0	0
0	2	17	32
0	22	42	69
0	27	47	94
0	35	60	125

3. 查询结果记录

4. 算法说明 (对应 solution_3.cpp)

3. 执行查询

1. 计算二维前缀和
2. 执行下一次查询
- 重置

原始矩阵 (v(i,j))

2	15	15	2
20	5	12	18
5	0	20	7
8	5	18	1

二维前缀和矩阵 (prefixsum[i][j])

公式: $prefixsum[i][j] = v[i][j] + prefixsum[i-1][j] + prefixsum[i][j-1] - prefixsum[i-1][j-1]$

0	0	0	0	
0	2	17	32	34
0	22	42	69	89
0	27	47	94	121
0	35	60	125	153

3. 查询结果记录

查询 1/2

参数: 起始坐标 (1, 1), 子矩阵大小 (2, 2), 右下角坐标 (2, 2)

子矩阵和: 42

计算过程: $prefixsum[2][2] - prefixsum[0][2] - prefixsum[2][0] + prefixsum[0][0] = 42 - 0 - 0 + 0 = 42$

