# 题目

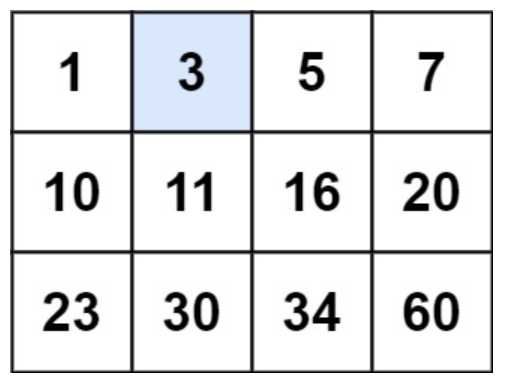
给你一个满足下述两条属性的m x n整数矩阵：

每行中的整数从左到右按非严格递增顺序排列。

每行的第一个整数大于前一行的最后一个整数。

给你一个整数 target ，如果 target 在矩阵中，返回 true ；否则，返回 false 。

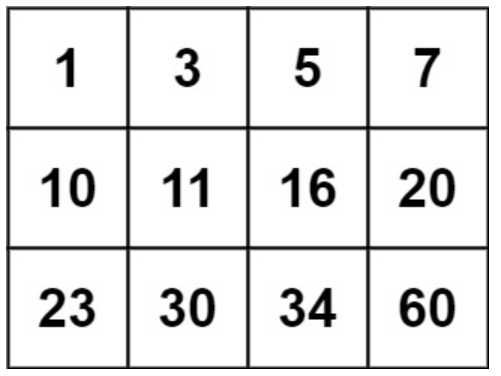
示例 1：



输入：matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

输出：true

示例 2：



输入：matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13

输出：false

提示：

m == matrix.length

n == matrix[i].length

1 <= m, n <= 100

-10^4 <= matrix[i][j], target <= 10^4

# 分析

## 方法一：模拟

思路：

代码：

class Solution {

public:

bool searchMatrix(vector<vector<int>>& matrix, int target) {

if (matrix.empty() || matrix[0].empty()) {

return false;

}

int m = matrix.size();

int n = matrix[0].size();

int row = 0;

int col = n - 1;

while (row < m && col >= 0) {

if (matrix[row][col] == target) {

return true;

} else if (matrix[row][col] > target) {

col--;

} else {

row++;

}

}

return false;

}

};

## 方法二：一次二分查找

思路：

若将矩阵每一行拼接在上一行的末尾，则会得到一个升序数组，我们可以在该数组上二分找到目标元素。

代码实现时，可以二分升序数组的下标，将其映射到原矩阵的行和列上。

这里是有序的二维数组，因此可以用二分查找。

代码：

class Solution {

public:

bool searchMatrix(vector<vector<int>>& matrix, int target) {

int m = matrix.size(), n = matrix[0].size(); // 获取行列数

int low = 0, high = m \* n - 1;

while (low <= high) {

int mid = (high - low) / 2 + low;

int x = matrix[mid / n][mid % n];

if (x < target) {

low = mid + 1;

} else if (x > target) {

high = mid - 1;

} else {

return true;

}

}

return false;

}

};

复杂度分析：

时间复杂度：(logmn)，其中m和n分别是矩阵的行数和列数。

空间复杂度：O(1)。