基本解法

Java代码

```
public class SortedMatrixSearch {
    public boolean searchMatrix(int[][] matrix, int target) {
       int m = matrix.length;if (m == 0 ) return false;//行数
       int n = matrix[0].length;if (n == 0) return false;// 列数
        //使用分治思想查找元素，矩阵区域需要左上角下标（行、列）、右下角下标（行、列）
        return searchSubMatrix(matrix, target, 0, 0, m - 1, n - 1);
    }

    public boolean searchSubMatrix(int[][] matrix, int target, int startRow,
int startColumn, int endRow, int endColumn) {
        //结束条件 元素个数小于1或者 矩阵最小元素大于目标值（该矩阵所有元素都大于目标值）
        if (startRow > endRow || startColumn > endColumn) return false;
        if (matrix[startRow][startColumn] > target) return false;
        //函数主功能：在对角线上查找元素 、分解问题、递归求解、合并
        //找到对角线右下角位置  细节问题：矩阵行、列可能不相等
        int diagonal_length = Math.min(endRow - startRow + 1, endColumn -
startColumn + 1);
        for (int i = 0; i < diagonal_length; i++) {
            //在对角线上找到则返回true
            if (matrix[startRow + i][startColumn + i] == target) return true;
            //如果到达对角线上最后一个元素 或者元素大于目标值，则进行分解-递归求解--合并
            if (i == diagonal_length - 1 || matrix[startRow + i + 1]
[startColumn + i + 1] > target) {
                //找到了分界点，寻找4个区域中剩下的两个 （右上、左下）
                return searchSubMatrix(matrix, target, startRow, startColumn +
i + 1, startRow + i, endColumn)
                        || searchSubMatrix(matrix, target, startRow + i + 1,
startColumn, endRow, startColumn + i);
            }
        }

        //等价关系式：f(m)=f(m1)||f(m2)  m1、m2为划分后待查找的子矩阵
        return false;
    }
}
```

优化解法

Java代码

```java
public boolean searchMatrix(int[][] matrix, int target) {
    int m = matrix.length;  if (m == 0) return false;//行数
    int n = matrix[0].length; if (n == 0) return false;// 列数
    //初始位置右上角元素
    int currentRow = 0; int currentColumn = n - 1;
    while (currentColumn >= 0 && currentRow < m) {
        //当前元素等于目标值，返回true
        if (matrix[currentRow][currentColumn] == target) return true;
        //若当前元素小于目标值 指针下移
        if (matrix[currentRow][currentColumn] < target) {
            currentRow++;
        } else {//若当前元素大于目标值 指针左移
            currentColumn--;
        }
    }
    return false;
}
```

Python代码

```python
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int):
        if not len(matrix) or not len(matrix[0]):
            return False
        row = 0
        col = len(matrix[0]) - 1
        while row != len(matrix) and col != -1:
            #排除当列，去左边部分找
            if matrix[row][col] > target:
                col -= 1
            #排除当行，去下边部分找
            elif matrix[row][col] < target:
                row += 1
            else:
                return True
        return False
```

C++代码

```cpp
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if(matrix.empty())  return false;
        //从右上角开始搜索
        int row = 0;
        int col = matrix.at(0).size() - 1;
```

```
        while (row <= matrix.size() - 1 && col >= 0){
            if (matrix.at(row).at(col) == target)  return true;
            else if(target < matrix.at(row).at(col)) //排除当列，去左边部分找
                col--;
            else if (target > matrix.at(row).at(col)) //排除当行，去下边部分找
                row++;
        }
        return false;
    }
};
```