拉勾教育
—互联网人实战大学—

分治算法：排序矩阵查找

中等/分治算法、矩阵、双指针

学习目标

拉勾教育
—互联网人实战大学—

- 理解分治思想的基本概念
- 掌握使用分治算法解决实际问题的流程
- 熟悉矩阵数据结构

题目描述

给定M×N矩阵，每一行、每一列都按升序排列，请编写代码找出某元素。

现有矩阵 matrix 如下：

```
[[  1,   4,   7, 11, 15],
  [  2,   5,   8, 12, 19],
  [  3,   6,   9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]]
```

给定 target = 5， 返回 true。

给定 target = 20，返回 false。

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

排序矩阵查找

- 输入一个整形矩阵，查找某元素是否在矩阵中出现
- 额外信息
  - 矩阵的每一行、每一列都按升序排列

matrix[i][j]<matrix[i+1][j]

matrix[i][j]<matrix[i][j+1]

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

```
[[ 1,  4,  7, 11, 15],
 [ 2,  5,  8, 12, 19],
 [ 3,  6,  9, 16, 22],
 [10, 13, 14, 17, 24],
 [18, 21, 23, 26, 30]]
```

二. Choose 数据结构及算法思维选择

数据结构选择

- 输入的数据类型 整形二维数组(矩阵）、整数（需要查找的目标）
- 输出的数据类型为一个布尔值，表示需要查找的整数是否在数组中
- 因为需要对矩阵进行处理，所以数据结构我们选择二维数组

二. Choose 数据结构及算法思维选择

算法思维选择

- 朴素算法：遍历数组中所有元素，查找目标是否存在
- 未利用额外信息：矩阵的每一行、每一列都按升序排列

```
public boolean searchMatrix(int[][] matrix, int target) {
    for(int i=0; i<matrix.length; i++)
        for(int j=0; j<matrix[0].length; j++)
            if(matrix[i][j] == target)
                return true;
    return false;
}
```

执行结果： 通过 显示详情 >

执行用时： 22 ms , 在所有 Java 提交中击败了 6.21% 的用户

内存消耗： 44.6 MB , 在所有 Java 提交中击败了 6.60% 的用户

二. Choose 数据结构及算法思维选择

算法思维选择

- 如果当前查找的元素比矩阵中的某个元素小

  那么我们可以排除这个元素右下方的所有元素

- 如果当前查找的元素比矩阵中的某个元素大

  那么我们可以排除这个元素左上方的所有元素

- 那么我们怎么利用这个线索呢？

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

target: 8

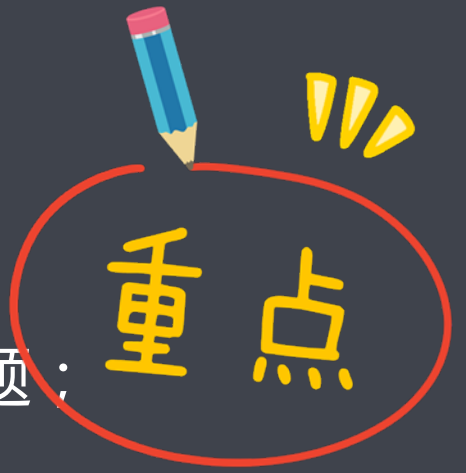
二. Choose 数据结构及算法思维选择

关键知识点：分治法（Divide-and-conquer）

- 把复杂的问题分成两个或更多的相同或相似的子问题，直到子问题可直接求解，原问题的解即子问题的解的合并
- 在计算机科学中，分治法一种很重要的算法范式
- 是很多高效算法的基础，如快速排序算法、快速傅立叶变换

二. Choose 数据结构及算法思维选择

关键知识点：分治法（Divide-and-conquer）



分治法解题的一般步骤：

（1）分解：将要解决的问题划分成若干规模较小的同类问题；

（2）求解：递归地求解各个子问题，当子问题划分得足够小时，用较简单的方法解决；

（3）合并：按原问题的要求，将子问题的解逐层合并构成原问题的解。

二. Choose 数据结构及算法思维选择

关键知识点：分治法（Divide-and-conquer）

回顾示例：快速排序

1	3	4	5	6	8	9	11	16

对示例数组进行排序，首先选择一个元素6

- 将数组中的元素划分为大于6和小于6的两部分
- 然后问题就变成了两个小数组的排序问题
- 我们可以继续对小数组采取分治策略，划分成更小的数组
- 当子数组中元素个数为1的时候无需排序

三. Code 基本解法及编码实现

解题思路剖析

使用分治法，先比较目标与矩阵对角线元素

找到元素返回true，找不到则分解

步骤一 分解：矩阵被划分成4个子问题

- 我们可以看到绿色部分一定小于目标元素
- 红色部分一定大于目标元素
- 我们只需在剩下的两个行列升序的矩阵中寻找目标元素

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

target: 8

三. Code 基本解法及编码实现

解题思路剖析

步骤二 子问题分别求解：

- 子问题元素个数为0或者最小元素大于目标返回false
- 否则对两个子矩阵继续递归分解

- 步骤三 合并：
  - 任一子矩阵内寻找到目标元素，则返回true

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

target:8

三. Code 基本解法及编码实现

复杂度分析

- 时间复杂度
  - 从矩阵左上方开始不断比较目标值与对角线元素，寻找划分位置
  - 如果未找到目标值，接着继续在两个子矩阵重复这个过程（递归）
  - 时间复杂度为O((m+n)*log(m+n))
- 空间复杂度
  - 需常数级临时变量
  - 递归调用占用额外空间，递归深度为log(m+n)
  - 因此空间复杂度为O(log(m+n))

三. Code 基本解法及编码实现

Java编码实现

执行结果： 通过 显示详情 >

执行用时： 7 ms ，在所有 Java 提交中击败了 46.64% 的用户

内存消耗： 44.2 MB ，在所有 Java 提交中击败了 67.48% 的用户

```
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
    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|| matrix[startRow][startColumn]>target)return false;
    int diagonal_length = Math.min(endRow - startRow + 1, endColumn - startColumn + 1);
    for (int i = 0; i < diagonal_length; i++) {//函数主功能：在对角线上查找元素 、分解问题、递归求解、合并
        if (matrix[startRow + i][startColumn + i] == target) return true;
        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);
        }
    }
    return false;
}
```

四. Consider 思考更优解

是否存在无效代码或者无效空间消耗

是否有更好的算法思维

- 能否找到一个更好的划分方式使得每次剩下的两个子矩阵其中之一为空？
- 经过观察，从矩阵左下角或右上角开始，可以达成上面的结果

四. Consider 思考更优解

算法思维选择

- 维护一个行指针、一个列指针
- 从右上角元素出发，比较目标元素与当前数值
- 15大于目标值，排除15右下方所有元素，指针左移
- 11大于目标值，排除11右下方所有元素， 指针左移
- 7小于目标值，排除7左上方所有元素，指针下移
- 找到目标元素8，返回true

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

target: 8

五. Code 最优解思路及编码实现

解题思路剖析

从矩阵右上方元素开始，比较当前元素与目标值的大小

- 若当前元素等于目标值，那么返回true
- 若当前元素小于目标值，那么当前元素左侧的元素都会小于目标值，指针下移
- 若当前元素大于目标值，那么当前元素右下方的元素都会大于目标值，指针左移
- 若指针在矩阵外，返回false

五. Code 最优解思路及编码实现

复杂度分析

- 时间复杂度
  - 最多比较m+n-1次，时间复杂度为O(m+n)
- 空间复杂度
  - 空间消耗方面只需常数级临时变量，空间复杂度为O(1)

五. Code 最优解思路及编码实现

Java编码实现

```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;
}
```

执行结果： 通过 显示详情 >

执行用时： 6 ms , 在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 44.4 MB , 在所有 Java 提交中击败了 28.11% 的用户

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

1. 若输入矩阵的每一行、每一列都按降序排列，算法应如何修改？
2. 若输入矩阵只有行是升序排序（列无序），算法应如何修改？

延伸扩展

- 分治算法是一种基础通用的算法思维
- 分治算法在实际工作中的应用
  - 大数据领域如MapReduce
  - 二分查找，快速排序，归并排序
  - 求数值的整数次方

$$x^{17} = x^{2^{2^{2^2}}} x$$

总结

- 理解分治思想的基本概念
- 掌握使用分治算法解决实际问题的流程
- 熟悉矩阵数据结构

课后练习

拉勾教育
—互联网人实战大学—

1. 多数元素（leetcode 169/简单）
2. 最大子序和（leetcode 53/简单）
3. 为运算表达式设计优先级（leetcode 241/中等）
4. 数组中的第K个最大元素（leetcode 215/中等）

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容