第1篇

> 知识点：查找 复习：线性表顺序查找、二分查找

二维数组的查找

题目描述

在一个二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

分析

一上来想到的方法是，先按第一列做查找，找到行首比该数字小的最大一行，就是该数字所在的行，然后按这一行做查找，找到该数字所在的列。这个想法是错误的，因为根本没有认真审题，没有理解题中给出的是什么样的数组。

`每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序`，数组可能是这样的：

```
1 2 8 9
2 4 9 12
4 7 10  13
6 8 11  15
```

在没有具体给出一个数组的情况下，光看文字着急去空想，错误的方法显然是把整个二维数组当作有序，也就是按从左到右从上到下顺序读一遍的有序。

正确的解法是，从某一个角开始查找，然后指针向两个方向移动。如果从左上角开始，由于current <target时，向右向下都更大，会有歧义（从右下角开始同理），如果从左下角开始，当current <target时，说明应当向右移动一格，当current>target时，说明应当向上移动一格，没有歧义。（从右上角开始同理）

代码如下：

```python
# -*- coding:utf-8 -*-
class Solution:
    def Find(self, target, array):
        # write code here
        if target is None or array is None:
            return False
        col = 0
        row = len(array)-1
        while row >= 0 and col < len(array[0]):
            if target == array[row][col]:
```

```
                return True
            elif target > array[row][col]:
                col += 1
            elif target < array[row][col]:
                row -= 1
        return False
```

查找算法

平均查找长度ASL：需和指定key进行比较的关键字的个数的期望值，称为查找算法在查找成功时的平均查找长度。

对于含有n个数据元素的查找表，查找成功的平均查找长度为：`ASL = Pi*Ci`的和。

Pi：查找表中第i个数据元素的概率。　　Ci：找到第i个数据元素时已经比较过的次数。

顺序查找

```
def Find(target, values):
    for i in range(0, len(values)):
        if values[i] == target:
            return True
    return False
```

ASL= 1/n * (1+2+3+...+n) = (n+1)/2 时间复杂度O(n)

二分查找

要求元素必须是有序的。 最坏的情况是查找到最后才找到，也就是经过了k次二分，n/(2^k)=1, n=2^k, k=log(n)，时间复杂度 O(log n)

通过循环实现：

```
def Binary(target, values):
    left = 0
    right = len(values) - 1
    while left <= right:
        mid = (left + right) // 2
        if values[mid] == target:
            return mid
        elif values[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

通过递归实现：

```python
def Binary_recursive(target, values, left, right):
    if left > right:
        return -1
    mid = (left + right) // 2
    if target == values[mid]:
        return mid
    elif target > values[mid]:
        return Binary_recursive(target, values, mid + 1, right)
    else:
        return Binary_recursive(target, values, left, mid - 1)
```