算法分析和复杂性理论

第 10 次作业

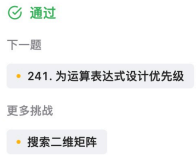
张瀚文 2201212865

1 搜索二维矩阵 II（240）

题目描述：编写一个高效的算法来搜索 m x n 矩阵 matrix 中的一个目标值 target 。该矩阵具有以下特性：
每行的元素从左到右升序排列。
每列的元素从上到下升序排列。

解题思路：由于矩阵 matrix 中每一行的元素都是升序排列的，因此我们可以对每一行都使用一次二分查找，判断 target 是否在该行中，从而判断 target 是否出现。

运行截图：



测试代码：

```
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) ->
bool:
        for row in matrix:
            idx = bisect.bisect_left(row, target)
            if idx < len(row) and row[idx] == target:
                return True
        return False
```

2 前 K 个高频元素（347）

题目描述：

给你一个整数数组 `nums` 和一个整数 `k` ，请你返回其中出现频率前 `k` 高的元素。你可以按 **任意顺序** 返回答案。

示例 1:

```
输入: nums = [1,1,1,2,2,3], k = 2
输出: [1,2]
```

示例 2:

```
输入: nums = [1], k = 1
输出: [1]
```

解题思路：
首先遍历整个数组，并使用哈希表记录每个数字出现的次数，并形成一个「出现次数数组」。找出原数组的前 k 个高频元素，就相当于找出「出现次数数组」的前 k 大的值。最简单的做法是给「出现次数数组」排序。但由于可能有 O(N) 个不同的出现次数（其中 N 为原数组长度），故总的算法复杂度不满足题目的要求。在这里，我们可以利用堆的思想：建立一个小顶堆，然后遍历「出现次数数组」：

如果堆的元素个数小于 k，就可以直接插入堆中。
如果堆的元素个数等于 k，则检查堆顶与当前出现次数的大小。如果堆顶更大，说明至少有 k 个数字的出现次数比当前值大，故舍弃当前值；否则，就弹出堆顶，并将当前值插入堆中。
遍历完成后，堆中的元素就代表了「出现次数数组」中前 k 大的值。

运行截图：

测试代码：

```
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        count = [(v,k) for k,v in collections.Counter(nums).items()] # 这里注意次数应该是放在前面的
        # 注意 Python 的 heap 是小根堆
        h = count[:k]
        heapq.heapify(h) # 初始化堆
```

```
        for i in range(k,len(count)):
            heapq.heappushpop(h,count[i])

        return [b for a,b in h ] # 选择前 k 个
```

3 猜数字大小（374）

题目描述：

猜数字游戏的规则如下：

- 每轮游戏，我都会从 **1** 到 ***n*** 随机选择一个数字。 请你猜选出的是哪个数字。
- 如果你猜错了，我会告诉你，你猜测的数字比我选出的数字是大了还是小了。

你可以通过调用一个预先定义好的接口 `int guess(int num)` 来获取猜测结果，返回值一共有 3 种可能的情况（`-1`，`1` 或 `0`）：

- -1：我选出的数字比你猜的数字小 `pick < num`
- 1：我选出的数字比你猜的数字大 `pick > num`
- 0：我选出的数字和你猜的数字一样。恭喜！你猜对了！ `pick == num`

返回我选出的数字。

解题思路：

我们可以使用二分查找来求出答案 pick。

运行截图：

测试代码：

```
class Solution:
    def guessNumber(self, n: int) -> int:
        left, right = 1, n
        while left < right:
            mid = (left + right) // 2
            if guess(mid) <= 0:
                right = mid   # 答案在区间 [left, mid] 中
            else:
                left = mid + 1   # 答案在区间 [mid+1, right] 中
```

```
    # 此时有 left == right，区间缩为一个点，即为答案
    return left
```