# 题目

给你一个下标从 0 开始的字符串数组 nums ，其中每个字符串 长度相等 且只包含数字。

再给你一个下标从 0 开始的二维整数数组 queries ，其中 queries[i] = [ki, trimi] 。对于每个 queries[i] ，你需要：

将 nums 中每个数字 裁剪 到剩下 最右边 trimi 个数位。

在裁剪过后的数字中，找到 nums 中第 ki 小数字对应的 下标 。如果两个裁剪后数字一样大，那么下标 更小 的数字视为更小的数字。

将 nums 中每个数字恢复到原本字符串。

请你返回一个长度与 queries 相等的数组 answer，其中 answer[i]是第 i 次查询的结果。

提示：

裁剪到剩下最右边 x 个数位的意思是不断删除最左边的数位，直到剩下 x 个数位。

nums 中的字符串可能会有前导 0 。

示例 1：

输入：nums = ["102","473","251","814"], queries = [[1,1],[2,3],[4,2],[1,2]]

输出：[2,2,1,0]

解释：

1、裁剪到只剩 1 个数位后，nums = ["2","3","1","4"] 。最小的数字是 1 ，下标为 2 。

2、裁剪到剩 3 个数位后，nums 没有变化。第 2 小的数字是 251 ，下标为 2 。

3、裁剪到剩 2 个数位后，nums = ["02","73","51","14"] 。第 4 小的数字是 73 ，下标为 1 。

4、裁剪到剩 2 个数位后，最小数字是 2 ，下标为 0 。

注意，裁剪后数字 "02" 值为 2 。

示例 2：

输入：nums = ["24","37","96","04"], queries = [[2,1],[2,2]]

输出：[3,0]

解释：

1、裁剪到剩 1 个数位，nums = ["4","7","6","4"] 。第 2 小的数字是 4 ，下标为 3 。

有两个 4 ，下标为 0 的 4 视为小于下标为 3 的 4 。

2、裁剪到剩 2 个数位，nums 不变。第二小的数字是 24 ，下标为 0 。

提示：

1 <= nums.length <= 100

1 <= nums[i].length <= 100

nums[i] 只包含数字。

所有 nums[i].length 的长度 相同 。

1 <= queries.length <= 100

queries[i].length == 2

1 <= ki <= nums.length

1 <= trimi <= nums[0].length

进阶：你能使用基数排序算法解决此问题吗？这种解法的复杂度又是多少？

# 分析

要解决这个问题，我们需要处理多个查询，每个查询要求将字符串数组中的每个数字裁剪到指定长度后，找到第k小的数字对应的原始下标。关键在于高效处理裁剪和排序，并保留原始下标信息。

思路

1、处理每个查询：对于每个查询[ki, trimi]，需要单独处理，因为裁剪长度不同，结果也不同。

2、裁剪数字：将每个数字字符串裁剪到最右边的trimi个字符。

3、排序裁剪后的数字：按照裁剪后的数字大小排序，若大小相同则按照原始下标从小到大排序。

4、获取结果：在排序后的列表中，取第ki-1个元素（因为下标从0开始）的原始下标作为该查询的结果。

代码：

class Solution {

public:

vector<int> smallestTrimmedNumbers(vector<string>& nums, vector<vector<int>>& queries) {

vector<int> answer;

int n = nums.size();

int len = nums[0].size(); // 所有数字的长度相同

for (auto& q : queries) {

int k = q[0];

int trim = q[1];

// 存储裁剪后的字符串和对应的原始下标

vector<pair<string, int>> trimmed;

for (int i = 0; i < n; ++i) {

// 裁剪到最右边的trim个字符

string t = nums[i].substr(len - trim, trim);

trimmed.emplace\_back(t, i);

}

// 排序：先按裁剪后的字符串比较，相同则按下标比较

sort(trimmed.begin(), trimmed.end(),

[](const pair<string, int>& a, const pair<string, int>& b) {

if (a.first != b.first) {

return a.first < b.first;

} else {

return a.second < b.second;

}

});

// 第k小的元素下标为k-1，取其原始下标

answer.push\_back(trimmed[k-1].second);

}

return answer;

}

};

解释：

1、裁剪处理：对于每个数字字符串，根据查询中的trimi，使用substr截取最右边的trimi个字符。例如，数字"102"裁剪1个字符后为"2"，裁剪2个字符后为"02"。

2、排序规则：排序时首先比较裁剪后的字符串（按字典序，由于都是数字且长度相同，字典序等价于数值大小），若字符串相同则比较原始下标，确保下标小的排在前面。

3、结果提取：排序后，第ki小的元素位于下标ki-1处，其对应的原始下标即为该查询的答案。

这种方法的时间复杂度为O(q \* n \* (t + n log n))，其中q是查询数量，n是数字个数，t是每个数字的长度。对于题目中n ≤ 100、q ≤ 100的约束，该方法高效且易于理解。

### 进阶思考（基数排序）

对于基数排序解法，可以按裁剪后的长度，从右往左逐位进行排序（因为裁剪后最右边的数字是最低位）。每次排序按当前位的数字分组，最后得到有序序列。其时间复杂度为`O(q \* (n \* t + t \* n)) = O(q \* n \* t)`，其中`t`是裁剪后的长度，在数字长度较小的情况下更具优势。