第二次小测题解

一、练习网站

第二次小测题目已添加到上次第一次小测发布的提单，链接如下：

2025 小测练习 - 题单 - 洛谷 | 计算机科学教育新生态

（提示：按住 ctrl 点击链接即可直接跳转到网站）

二、题目一：虚拟系统路径分析

1. 思路一：横向比较

通过题目，可以分析出公共前缀一定是所有路径的交集。我们可以直接假设第一条路径就是公共前缀，然后拿它依次与后面的每一条路径进行比对。在比对过程中，一旦发现某一层级不匹配，就对当前的公共前缀做“减法”（截断），只保留相同的部分。随着遍历的进行，公共前缀会越来越短或保持不变，遍历结束后的结果即为最终答案。

注意题目中的提示：通过 path = input().rstrip()读取一整行输入

```
# 记录路径总数 n
n = int(input())
paths = []
# 记录所有路径
# 对于这样一个路径"Path1::Path2::Path3"
# 在 path 变量中是['Path1', 'Path2', 'Path3']
"""
paths 中存储的是：
[
['Path11', 'Path12', 'Path13'],
['Path21', 'Path22', 'Path23'],
['Path31', 'Path32', 'Path33'],
]
"""
for i in range(n):
    path = input().rstrip().split("::")
    paths.append(path)

# 由于公共路径一定是所有路径的子集，一定是每一条路径的一部分
# 所以直接把第一条路径设为公共路径，对它做减法，去掉不公共的部分
common_path = paths[0]
# 比较公共路径和剩下 n-1 条路径
for i in range(1,n):
    # 首先确定较短的路径的长度，防止列表越界
    num = min(len(common_path),len(paths[i]))
    common_path = common_path[:num]
    # 在这个循环中，会一级一级比较路径的每一部分
    for j in range(num):
```

```
        # 如果从第 j 部分开始不相同，那 j 之后的部分不需要比较了
        if common_path[j] != paths[i][j]:
            # 使用切片保存公共部分
            common_path = common_path[0:j]
            # break 跳出循环，实现上面的“j 之后的部分不需要比较”
            break

# 新建一个空字符串，用来保存公共路径
root = ""
# common_path 是一个列表
for path in common_path:
    # 把每一部分加到 root 中，后面加上 ::
    root += path
    root += "::"
print(root)
```

2. 思路二：纵向比较

注意到题目要求的是层级匹配（以::为单位）而非简单的字符匹配。我们可以将所有路径想象成一个左对齐的二维矩阵。对于公共前缀，只需要从第一列开始，逐列（逐层）检查所有路径的该层是否相同。一旦发现某一列有不一致的元素，公共路径即在此终止。

在确定比较边界时，可以通过遍历所有路径列表，找出长度最短的路径的长度。这样可以确定后续循环的安全范围，避免数组越界。

```
line = input().rstrip() # 读取第一行
n = int(line)

paths = []
for i in range(n):
    # 读取每一行路径
    p = input().rstrip()
    paths.append(p)

# 将每个路径字符串按 "::" 切割成列表
split_paths = []
for p in paths:
    parts = p.split("::")
    split_paths.append(parts)

# 找出所有路径中，层级最少的那个路径的长度
# 因为公共前缀的长度不可能超过最短的那条路径
# 假设第一个路径是最短的，然后遍历剩下的去比较
min_len = len(split_paths[0])
for parts in split_paths:
    if len(parts) < min_len:
```

```
        min_len = len(parts)

# 纵向扫描
common_layers = []

for i in range(min_len):
    # 以第一个路径的第 i 层作为 standard
    standard = split_paths[0][i]

    # 设置一个标记，假设这一层都一样
    is_same = True

    # 内层循环：检查其他所有路径的第 i 层
    for j in range(n):
        # 如果发现有一个路径的第 i 层和 standard 不一样
        if split_paths[j][i] != standard:
            is_same = False
            break # 只要发现一个不一样，直接跳出内层循环

    # 检查标记
    if is_same == True:
        # 如果都一样，就把这一层加到公共列表里
        common_layers.append(standard)
    else:
        # 如果不一样，跳出外层循环
        break

# 拼接结果输出
result = ""
for layer in common_layers:
    # 拼接层级名和分隔符
    result = result + layer + "::"

print(result)
```

3. 思路三：排序

排序解法的主要思路是利用字符串排序的特性，将n个路径的比较简化为只比较两个路径。也就是说，当一组字符串排好序后，它们是按照字符编码大小排列的，那么拥有相同前缀的字符串会紧挨在一起，差异最大的两个字符串一定分别位于列表的第一个和最后一个。如果这一头一尾两个路径都有公共前缀，那么排在它们中间的所有路径一定也拥有这个公共前缀。因此，我们只需要比较这两个路径即可。

```
n = int(input())
paths = []
```

```python
for _ in range(n):
    paths.append(input().rstrip())

# 排序,字符串排序是按字符 ASCII 码比的，差异最大的会被放置到两头
paths.sort()

# 只拿第一个和最后一个比较
first_path = paths[0].split("::")
last_path = paths[-1].split("::")

# 找出这两个路径的公共层级
common = []
# 为了防止越界，循环长度取两者的较小值
min_len = min(len(first_path), len(last_path))

for i in range(min_len):
    if first_path[i] == last_path[i]:
        common.append(first_path[i])
    else:
        break

# 拼接输出
result = ""
for item in common:
    result += item + "::"

print(result)
```

二、题目二：寻找第n大元素

1. 思路一：全局排序，*sort*()方法【100 分】

要找到第n大的数，最简单的办法就是把所有人都排好队。虽然题目给的是无序数组，但如果我们调用列表的sort()函数，按从小到大（升序）排列，那么最大的数就在最后一个（索引为−1）。第 2 大的数就在倒数第二个（索引为−2）。同理，第 n 大的数就在倒数第n个位置（索引为−n）。

这个代码的空间复杂度是 O(L)。最主要的部分是在代码中定义了一个空列表*list*1，然后用一个循环，把输入的L个整数全部读进去，存到这个列表里用了O(L)的空间，内存占用会随着输入量L的增加而成正比增加。

```python
# 思路：把列表存下来，用 sort()方法排序
# 输出倒数第 n 个数
n = int(input().rstrip())
l = int(input().rstrip())
list1 = []
for i in range(l):
    list1.append(int(input().rstrip()))
```

```
list1.sort()
print(list1[-n])
```

2. 思路二：维护固定长度有序列表【120 分】

本题第六个测试数据长度范围为$1 \le n \le 10 \le len(list) \le 5 \times 10^5$，而且在题目中提示第六个测试数据有内存空间限制，同时题目右上角也有如图 1 所示的提示，这些均提示此题对内存有限制。此外，某种情况下洛谷测试时会出现“MLE”，则表示超出内存。

图 1 内存限制提示

解决思路是要找第n个最大的数，只需要维护一个长度为n的列表，并不需要存储所有列表。这样的话，空间复杂度为O(n)（题目中给定数据范围$n \le 10$）。

```
n = int(input().rstrip())
l = int(input().rstrip())
list1 = []
# 把前 n 个数存到列表中
for i in range(n):
    list1.append(int(input().rstrip()))
# 把列表排序一下
list1.sort()
# 只需要把剩下的 l-n 个数字逐个放入列表中
# 排序完后去掉最小的元素
# 保证列表长度一直维持在 n 个
l -= n
for i in range(l):
    list1.append(int(input().rstrip()))
    list1.sort()
    list1.pop(0)
# 第一个元素就是最小的元素，输出
print(list1[0])
```

或

```
n = int(input().rstrip())
l = int(input().rstrip())
list1 = []
# 把前 n 个数存到列表中
for i in range(n):
    list1.append(int(input().rstrip()))

# 只需要把剩下的 l-n 个数字逐个与列表中最小的元素进行比较
# 保证列表长度一直维持在 n 个
```

```
l -= n
for i in range(l):
    min_num = min(list1)
    x = int(input().rstrip())
    if x > min_num:
        list1.remove(min_num)
        list1.append(x)

print(min(list1)) # 列表中最小的元素就是第 n 大的元素

```