AI1804 算法设计与分析
第 3 次课上练习：哈希表与优先队列

练习说明：

- 本次练习共 2 道题，总时间 90 分钟（两节课）
- 重点：优先队列（priority queue）的实现与应用
- 为图搜索算法（下次课）做准备
- 每道题都需要用 Python 编程实现

问题 3-1：字符串基数排序

问题描述

给定一个字符串数组，所有字符串长度相同。请使用基数排序的思想对它们排序。

示例：

```
输入：["abc", "abd", "aba", "aaa", "bcd"]
输出：["aaa", "aba", "abc", "abd", "bcd"]

输入：["dog", "cat", "bat", "rat"]
输出：["bat", "cat", "dog", "rat"]
```

要求：

- 从最后一个字符到第一个字符，依次排序
- 每一轮使用计数排序（利用字符只有 256 种的特点）
- 必须保证稳定性
- 时间复杂度：$O(d \cdot n)$，d 是字符串长度

代码框架

```
def counting_sort_by_char(arr, char_index):
    """
    按指定字符位置进行稳定的计数排序

    参数:
        arr: 字符串数组
        char_index: 要排序的字符位置（0表示第一个字符）

    返回:
        排序后的数组
    """
```

```
    # TODO: 实现按字符位置的计数排序
    # 提示: 使用计数数组（256个桶）+ 累积和 + 稳定填充
    pass

def string_radix_sort(arr):
    """
    字符串基数排序

    参数:
        arr: 等长字符串数组

    返回:
        排序后的数组

    时间复杂度: O(d * n), d是字符串长度
    """
    if not arr:
        return []

    # TODO: 从最后一个字符到第一个字符，依次调用counting_sort_by_char

    pass

# 测试代码
arr1 = ["abc", "abd", "aba", "aaa", "bcd"]
sorted_arr1 = string_radix_sort(arr1)
print(f"排序结果: {sorted_arr1}")
# 应输出 ['aaa', 'aba', 'abc', 'abd', 'bcd']

arr2 = ["dog", "cat", "bat", "rat"]
sorted_arr2 = string_radix_sort(arr2)
print(f"排序结果: {sorted_arr2}")
# 应输出 ['bat', 'cat', 'dog', 'rat']
```

问题 **3-2**：**K** 路归并与去重

问题描述

某推荐系统需要从 K 个不同的推荐算法中合并结果，找出总体得分最高的 N 个物品。

输入：

- K 个未排序的列表，每个列表包含 `(item_id, priority)` 对
- 同一 `item_id` 可能在多个列表中出现，需要保留最大 **priority**

输出：

- 前 N 个得分最高的不同 item，按 priority 降序排列
- 每个 item 只出现一次（取最大 priority）

示例：

输入：

```
lists = [
  [("A",100), ("C",60), ("B",80)],    # 未排序
  [("E",50), ("A",90), ("D",85)],     # A重复，priority更小
  [("F",40), ("B",95), ("C",70)]      # B和C重复
]
n = 3
```

输出：`[("A",100), ("B",95), ("D",85)]`

解释：

```
- A在list0和list1中出现，保留最大值100
- B在list0和list2中出现，保留最大值95
- D只在list1，priority=85
- 按priority降序排列
```

算法要求：

- 时间复杂度：$O(N \log K)$，其中 N 是总元素数，K 是列表个数
- 空间复杂度：$O(N)$

代码框架

```
import heapq

def merge_k_unsorted_lists_top_n(lists, n):
    """
    合并K个未排序列表，获取Top-N

    参数:
        lists: K个未排序列表，[(item_id, priority), ...]
        n: 返回前n个item
```

```
    返回:
        [(item_id, priority), ...] 前n个, 按priority降序

    时间复杂度要求: O(N log K)
    """
    if not lists or n <= 0:
        return []

    # TODO: 实现算法

    return []

# ========== 测试代码 ==========

print("=== 测试1: K个未排序列表, 获取Top-3 ===")
# 注意: 每个列表都是未排序的!
lists1 = [
    [("A", 100), ("C", 60), ("B", 80)],      # 未排序
    [("E", 50), ("A", 90), ("D", 85)],       # 未排序
    [("F", 40), ("B", 95), ("C", 70)]        # 未排序
]
result1 = merge_k_unsorted_lists_top_n(lists1, 3)
print(f"Top-3结果: {result1}")

# 验证:
# A的最大priority: 100 (出现在list0和list1)
# B的最大priority: 95 (出现在list0和list2)
# D的最大priority: 85 (出现在list1)
assert len(result1) == 3
assert result1[0] == ("A", 100)
assert result1[1] == ("B", 95)
assert result1[2] == ("D", 85)
print("  Top-3测试通过")

print("\n=== 测试2: 键冲突保留最大值 ===")
lists2 = [
    [("X", 50), ("Y", 30)],
    [("X", 100), ("Z", 20)],  # X出现, 更大priority
    [("X", 30), ("W", 40)]     # X再次出现, 更小priority
]
result2 = merge_k_unsorted_lists_top_n(lists2, 2)
print(f"Top-2结果: {result2}")

# X应该保留最大的100
assert result2[0] == ("X", 100)
print("  键冲突测试通过")

print("\n=== 测试3: n大于总item数 ===")
```

```
lists3 = [
    [("A", 100)],
    [("B", 90)]
]
result3 = merge_k_unsorted_lists_top_n(lists3, 10)
assert len(result3) == 2  # 只有2个不同item
print("  边界测试通过")

print("\n=== 测试4: 包含空列表 ===")
lists4 = [
    [("A", 100), ("B", 80)],
    [],  # 空列表
    [("C", 90), ("A", 70)]  # A重复，保留100
]
result4 = merge_k_unsorted_lists_top_n(lists4, 3)
assert len(result4) == 3
assert ("A", 100) in result4
print("  空列表测试通过")

print("\n" + "="*50)
print("所有测试通过！")
print("="*50)
```

提交说明

- 将代码保存在单个 Python 文件中
- 确保所有测试用例通过
- 代码应有适当注释
- 分析时间复杂度