4 第二讲 作业记录（二）

笔记本： 浙江大学《数据结构》
创建时间： 2025/3/18 22:34 **更新时间：** 2025/4/6 16:13
作者： panhengye@163.com
URL： https://pintia.cn/problem-sets/1873565885118418944/exam/problems/type/7?...

题目：02-线性结构3 Reversing Linked List

提交结果

题目	用户	提交时间
02-线性结构3	飞翔的小师弟	2025/03/18 22:03:03
编译器	**内存**	**用时**
Python (python3)	37500 / 65536 KB	307 / 400 ms
状态	**分数**	**评测时间**
答案正确	25 / 25	2025/03/18 22:03:03

评测详情

测试点	提示	内存(KB)	用时(ms)	结果	得分
0	sample 有尾巴不反转, 地址取上下界	2984	15	答案正确	12 / 12
1	正好全反转	3060	14	答案正确	3 / 3
2	K=N全反转	3064	14	答案正确	2 / 2
3	K=1不用反转	3020	14	答案正确	2 / 2
4	N=1 最小case	3280	14	答案正确	2 / 2
5	最大N,最后剩K-1不反转	37500	307	答案正确	3 / 3
6	有多余结点不在链表上	3068	14	答案正确	1 / 1

【感悟】

1. **针对c语言中的数组、链表数据类型，可以用python中的列表和字典类型来应对。**对本题而言，链表可以用{addr: [data, next_addr]}的方式实现
2. 在设计算法的时候要特别注意边界，比如本题中获取数据的第一行、输出数据的最后一行都要特殊处理
3. 当遇到“按照固定步长切片，不满足步长的部分不变”的问题时，可以巧妙利用range函数和%实现
4. 列表、字符串、元组类型的反转，都可以利用[::-1]这种特殊的切片方式实现，即针整个序列（参数忽略开头和结尾）倒着走
5. 为了代码的健壮性，建议在关键位置做检查。最常见的就是检查为空这种特例

这些技巧不仅适用于这道题，也适用于许多其他算法问题。特别是将复杂数据结构映射到Python内置类型的能力，以及利用Python特有语法（如切片）简化代码的思路，都是解决算法问题的宝贵技能。

【代码】

```
def get_data():

    fisrt_addr, N, K = input().split() # 获取第一行数据，即首地址、节点总数和组数
    N, K = int(N), int(K)

    nodes = {}

    # 从第二行开始的数据结构为Address Data Next，所以用循环处理
    for _ in range(N):
        addr, data, next_addr = input().split()
        nodes[addr] = [data, next_addr]

    return fisrt_addr, nodes, K

def build_list(fisrt_addr, nodes):

    linked_list = []
    addr = fisrt_addr

    while addr != "-1": # 题目中用-1表示null
        linked_list.append(addr)
        addr = nodes[addr][1] # 获取下一个节点的地址
    return linked_list

def reverse_list(linked_list, K):

    for i in range(0, len(linked_list) - len(linked_list) % K, K):
        linked_list[i:i+K] = linked_list[i:i+K][::-1]

    return linked_list

def print_list(linked_list, nodes):

    # 检查链表是否为空
    if not linked_list:
        print("-1")
        return

    for i in range(len(linked_list) -1):
        print(f"{linked_list[i]} {nodes[linked_list[i]][0]} {linked_list[i+1]}")

    print(f"{linked_list[-1]} {nodes[linked_list[-1]][0]} -1") # 按照题目要求，最后一个节点输出后没有下一个节点，输出-1表示指针指向null

def main():

    # 读取数据
    fisrt_addr, nodes, K = get_data()

    # 构建链表
    linked_list = build_list(fisrt_addr, nodes)

    # 分组反转
    linked_list = reverse_list(linked_list, K)

    # 输出结果
    print_list(linked_list, nodes)

if __name__ == "__main__":
    main()
```

【整体思路】

1. **读取输入**：

- 首先读取链表头节点地址、节点总数 `N` 和每组的长度 `K`。
- 然后循环读取每个节点的地址、数据和下一个节点的地址，将它们存储在字典 `nodes` 中。

2. **构建链表**：
   - 从链表头节点开始，通过不断访问下一个节点的地址，将链表中的节点地址按顺序存储在列表 `linked_list` 中。
3. **分组反转**：
   - 使用切片操作，每 `K` 个元素为一组进行反转。如果链表节点数不是 `K` 的整数倍，最后一组不进行反转。
4. **输出结果**：
   - 按照题目要求的格式输出处理后的链表，最后一个节点的下一个地址为 `-1`。

【关键点解析1】

```
linked_list[i:i + K][::-1]
```

代码中包含了两次切片操作，其具体作用如下：

第一次切片 `linked_list[i:i + K]`

这部分切片操作的作用是从列表`linked_list`中提取出一个子列表。其中：

- `i` 是切片的起始索引，表示从`linked_list`的第`i`个元素开始提取（索引从 0 开始）。
- `i + K` 是切片的结束索引，表示提取到第`i + K`个元素之前的所有元素（不包含索引为`i + K`的元素）。

也就是说，通过`linked_list[i:i + K]`会得到`linked_list`中从索引`i`开始，长度为`K`的一个子列表片段。比如`linked_list = [1, 2, 3, 4, 5, 6]`，当`i = 2, K = 3`时，`linked_list[2:2 + 3]` 得到的子列表就是`[3, 4, 5]`。

第二次切片 `[::-1]`

这是一个特殊的切片语法，省略了起始和结束索引，只指定了步长为`-1` 。它的作用是将前面得到的子列表进行反转。例如对于子列表`[3, 4, 5]` ，经过`[::-1]`操作后，会得到`[5, 4, 3]` 。

【关键点解析2】

```
for i in range(0, len(linked_list) - len(linked_list) % K, K)
```

`range()`函数一般有三种形式：

- range(stop)：生成从 0 开始到stop - 1的整数序列。
- range(start, stop)：生成从start开始到stop - 1的整数序列。
- range(start, stop, step)：生成从start开始，按照step的步长递增，直到小于stop的整数序列。

在for i in range(0, len(linked_list) - len(linked_list) % K, K):这行代码中，使用的是第三种形式：

> 针对range(起点，终点，步长)这种形式，可以类比厨师切黄瓜：可以掐头去尾；可以控制切的厚薄，决定是切片还是切段

- 0 是start，表示迭代从 0 开始。
- len(linked_list) - len(linked_list) % K 是stop。其中len(linked_list) % K计算的是linked_list的长度除以K后的余数，len(linked_list) - len(linked_list) % K的结果是不超过linked_list长度且为K的最大整数倍的位置索引。这样可以确保只对完整的K个元素一组进行操作，最后不完整的一组不处理。
- K 是step，表示每次迭代的步长为K，即每次循环i的值增加K 。

假设 linked_list = [1, 2, 3, 4, 5, 6, 7, 8]，K = 3。

1. 首先计算 len(linked_list) 的值，len(linked_list) 即列表 linked_list 的长度，这里 len(linked_list) = 8。
2. 接着计算 len(linked_list) % K 的值，8 % 3 = 2（% 是取余运算符）。
3. 然后计算 len(linked_list) - len(linked_list) % K 的值，即 8 - 2 = 6。
4. 此时 range(0, len(linked_list) - len(linked_list) % K, K) 就变成了 range(0, 6, 3)。

根据 range() 函数的规则，它会生成一个整数序列，从 0 开始，以步长 3 递增，直到小于 6 的整数序列，也就是 [0, 3]。

接下来 for 循环的执行过程如下：

- 第一次循环：
  - i = 0，此时可以对 linked_list[0:0 + 3]（即 linked_list[0:3]，对应列表中的 [1, 2, 3]）这一组元素进行后续操作（比如反转）。
- 第二次循环：
  - i = 3，此时可以对 linked_list[3:3 + 3]（即 linked_list[3:6]，对应列表中的 [4, 5, 6]）这一组元素进行后续操作（比如反转）。

当 i 再次递增为 6 时，循环条件 i < 6 不满足，循环结束，而列表中剩余的 [7, 8] 这两个元素（不足 K = 3 个）就不会被处理，符合我们只对完整

的 K 个元素一组进行操作的要求。