1 第一讲 作业记录

笔记本: 浙江大学《数据结构》
创建时间: 2025/3/16 12:16 **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?...

01-复杂度1 最大子列和问题

提交结果

题目	用户	提交时间
01-复杂度1	飞翔的小师弟	2025/03/16 11:37:29
编译器	内存	用时
Python (python3)	11116 / 65536 KB	78 / 10000 ms
状态	分数	评测时间
答案正确	20 / 20	2025/03/16 11:37:30

评测详情

测试点	提示	内存(KB)	用时(ms)	结果	得分
0	sample 有正负，负数开头结尾，最大和有更新	2760	14	答案正确	12 / 12
1	100个随机数	3004	14	答案正确	2 / 2
2	1000个随机数	2812	15	答案正确	2 / 2
3	10000个随机数	3416	20	答案正确	2 / 2
4	100000个随机数	11116	78	答案正确	2 / 2

```
def max_subarray_sum(n, array):

    current_sum = 0
    max_sum = 0

    for i in range(n):

        # 决定是继续当前子列还是以新元素重新开始
        current_sum = max(array[i], current_sum + array[i])

        # 更新最大子列和
        max_sum = max(max_sum, current_sum)

    return max_sum if max_sum > 0 else 0 # 如果结果为负，则直接返回0

n = int(input())  # 将输入转换为整数
array = list(map(int, input().split()))  # 将输入字符串分割并转换为整数列表

print(max_subarray_sum(n, array))
```

01-复杂度2 Maximum Subsequence Sum

提交结果

题目	用户	提交时间
01-复杂度2	飞翔的小师弟	2025/03/16 12:37:19
编译器	内存	用时
Python (python3)	3560 / 65536 KB	19 / 200 ms
状态	分数	评测时间
答案正确	25 / 25	2025/03/16 12:37:19

评测详情

测试点	提示	内存(KB)	用时(ms)	结果	得分
0	sample换1个数字。有正负，负数开头结尾，有并列最大和	3012	15	答案正确	13 / 13
1	最大和序列中有负数	3012	15	答案正确	2 / 2
2	并列和对应相同i但是不同j，即尾是0	3012	15	答案正确	2 / 2
3	1个正数	2856	15	答案正确	2 / 2
4	全是负数	2996	15	答案正确	2 / 2
5	负数和0	2884	15	答案正确	2 / 2
6	最大和前面有一段是0	3044	15	答案正确	1 / 1
7	最大N	3560	19	答案正确	1 / 1

思路：

- 当决定开始新的子序列时（current_sum = array[i]），记录新的起始位置
- 当更新最大和时，同时更新最大子序列的起始和结束位置
- 特别处理全负数的情况

第一版

```
def max_subarray_sum(n, array):

    current_sum = 0
    max_sum = 0
    start = 0
    end = 0

    for i in range(n):

        # 决定是继续当前子列还是开始新的子列
        if array[i] > current_sum + array[i]:
            current_sum = array[i]
            start = i  # 记录新子序列的起始位置
        else:
            current_sum = current_sum + array[i]

        # 如果最大子列和更新，则更新结束位置
        if current_sum > max_sum:
            max_sum = current_sum
```

```
            end = i

    # 当所有数字都是负数时，最大和定义为0，输出整个序列的第一个数字和最后一个数字
    if max_sum <= 0:
        return 0, array[0], array[-1]
    else:
        return max_sum, start, end

array = [-10, 1, 2, 3, 4, -5, -23, 3, 7, -21]
n = 10

max_sum, start, end = max_subarray_sum(n, array)
print(f"{max_sum} {array[start]} {array[end]}")
```

结果发现起始位置不对。这是因为逻辑存在不严谨，实际上当开始新子列时要记录的起始位置，只有在当最大子列和更新的时候才需要记录起点和终点。修改后的代码如下：

```
def max_subarray_sum(n, array):

    current_sum = 0
    max_sum = 0
    start = 0
    end = 0
    temp_start = 0  # 添加临时起始位置变量

    for i in range(n):

        # 决定是继续当前子列还是开始新的子列
        if array[i] > current_sum + array[i]:
            current_sum = array[i]
            temp_start = i  # 记录新子序列的起始位置
        else:
            current_sum = current_sum + array[i]

        # 如果最大子列和更新，则更新结束位置和起始位置
        if current_sum > max_sum:
            max_sum = current_sum
            end = i
            start = temp_start  # 更新最大子序列的起始位置

    # 当所有数字都是负数时，最大和定义为0，输出整个序列的第一个数字和最后一个数字
    if max_sum <= 0:
        return 0, array[0], array[-1]
    else:
        return max_sum, start, end

array = [-10, -1, -2, -3, -4, -5, -23, -3, -7, -21]
n = 10

max_sum, start, end = max_subarray_sum(n, array)
print(f"{max_sum} {array[start]} {array[end]}")
```

- 添加临时起始位置变量 temp_start
- 在决定开始新子序列时更新 temp_start
- 只有在更新最大和时才更新最终的 start 和 end
- 修改判断条件，使逻辑更清晰

评测详情

测试点	提示	内存(KB)	用时(ms)	结果	得分
0	sample换1个数字。有正负，负数开头结尾，有并列最大和	2956	15	答案正确	13 / 13
1	最大和序列中有负数	2904	14	答案正确	2 / 2
2	并列和对应相同但是不同，即尾是0	2836	14	答案正确	2 / 2
3	1个正数	2856	18	答案正确	2 / 2
4	全是负数	3020	15	答案错误	0 / 2
5	负数和0	3060	14	答案错误	0 / 2
6	最大和前面有一段是0	3056	15	答案正确	1 / 1
7	最大N	3380	18	答案正确	1 / 1

提交时发现，其中测试集4、5没有通过。这是什么原因呢?

在全负数情况下:

1. current_sum 会在每次迭代后变为负数
2. max_sum 保持为初始值 0
3. 条件 current_sum > max_sum 永远不会满足
4. 因此 start 和 end 可能没有被正确初始化

再次修改

```
def max_subarray_sum(n, array):


    current_sum = array[0]
    max_sum = array[0]
    start = 0
    end = 0
    temp_start = 0


    for i in range(1, n):

        # 决定是继续当前子列还是开始新的子列
        if array[i] > current_sum + array[i]:
            current_sum = array[i]
            temp_start = i
        else:
            current_sum = current_sum + array[i]


        # 如果最大子列和更新，则更新结束位置和起始位置
        if current_sum > max_sum:
            max_sum = current_sum
            end = i
            start = temp_start


    # 当所有数字都是负数时，最大和定义为0，输出整个序列的第一个数字和最后一个数字
    if max_sum <= 0:
        return 0, array[0], array[n-1]
    else:
        return max_sum, array[start], array[end]  # 返回实际元素值而非索引


array = [-10, 0, -2, -3, 0, -5, -23, 0, -7, -21]
n = 10


max_sum, start, end = max_subarray_sum(n, array)
print(f"{max_sum} {start} {end}")
```

大部分测试都通过了，但测试点5（负数和0）通不过:

评测详情

测试点	提示	内存(KB)	用时(ms)	结果	得分
0	sample换1个数字，有正负，负数开头结尾，有并列最大和	3028	15	答案正确	13 / 13
1	最大和序列中有负数	2872	14	答案正确	2 / 2
2	并列和对应相同(但是不同)，即尾是0	2992	14	答案正确	2 / 2
3	1个正数	2824	14	答案正确	2 / 2
4	全是负数	3036	14	答案正确	2 / 2
5	负数和0	2948	14	答案错误	0 / 2
6	最大和前面有一段是0	2844	14	答案正确	1 / 1
7	最大N	3628	17	答案正确	1 / 1

最终答案：

```
def max_subarray_sum(n, array):
    current_sum = 0
    max_sum = -float('inf')  # 初始化为负无穷大
    start = 0
    end = n-1  # 默认为整个数组
    temp_start = 0


    # 记录第一个0的位置（如果存在）
    first_zero = -1
    for i in range(n):
        if array[i] == 0:
            first_zero = i
            break


    for i in range(n):
        # 决定是继续当前子列还是开始新的子列
        if current_sum + array[i] < 0:
            current_sum = 0
            temp_start = i + 1  # 下一个位置开始新的子序列
        else:
            current_sum += array[i]

            # 如果当前和大于最大和，或者当前和等于最大和但起始位置更靠前
            if (current_sum > max_sum) or (current_sum == max_sum and temp_start
< start):
                max_sum = current_sum
                start = temp_start
                end = i

    # 如果最大和仍为负无穷大，说明全是负数
    if max_sum < 0:
        return 0, array[0], array[n-1]

    # 如果最大和为0，且存在0元素，返回第一个0
    if max_sum == 0 and first_zero != -1:
        return 0, array[first_zero], array[first_zero]

    return max_sum, array[start], array[end]


n = int(input())
array = list(map(int, input().split())) # 将输入字符串分割并转换为整数列表

max_sum, start, end = max_subarray_sum(n, array)
print(f"{max_sum} {start} {end}")
```