动态规划问题：求数列中连续子数列的最大和

题干

动态规划问题

无后效性是指如果在**某个阶段**上过程的状态已知，则

1. 从**此阶段以后**过程的发展变化仅与**此阶段**的状态**有关**;
2. 与过程在**此阶段以前**的阶段所经历过的状态**无关**;

利用动态规划方法求解多阶段决策过程问题，过程的状态必须具备**无后效性。**

解题步骤

把实例拆分成子问题

$$(a_n) = (-2, 1, -3, 4, -1, 2, 1, -5, 4)$$

大问题：在数列中找到连续子数列的最大和。

我们可能得到如下子问题：

1. 经过 -2 的连续子数列的最大和是多少;
2. 经过 1 的连续子数列的最大和是多少;
3. 经过 -3 的连续子数列的最大和是多少;
4. 经过 4 的连续子数列的最大和是多少;
5. 经过 -1 的连续子数列的最大和是多少;
6. 经过 2 的连续子数列的最大和是多少;
7. 经过 1 的连续子数列的最大和是多少;
8. 经过 -5 的连续子数列的最大和是多少;
9. 经过 4 的连续子数列的最大和是多少;

这又带来了很强的不确定性，例如：
经过1（第二个数字）的连续子数列有

$$(-2, 1)(1)(1, -3)(1, -3, 4)$$

不确定性导致以下问题：

1. 子问题“有后效性”：上一个子问题和下一个子问题之间的联系根本看不出来。
2. 子问题自身也很难解出来。

怎样解决不确定性？ —— 严格定义子问题，限定更精确的范围，借此找到联系

现在加上限定词 **从头开始到结尾是 必须包含结尾**

1. **从头开始到结尾是** -2 的连续子数列(**必须包含结尾**)的最大和是多少;
2. **从头开始到结尾是** 1 的连续子数列(**必须包含结尾**)的最大和是多少;
3. **从头开始到结尾是** -3 的连续子数列(**必须包含结尾**)的最大和是多少;

4. **从头开始到结尾是** 4 的连续子数列(**必须包含结尾**)的最大和是多少;
5. **从头开始到结尾是** -1 的连续子数列(**必须包含结尾**)的最大和是多少;
6. **从头开始到结尾是** 2 的连续子数列(**必须包含结尾**)的最大和是多少;
7. **从头开始到结尾是** 1 的连续子数列(**必须包含结尾**)的最大和是多少;
8. **从头开始到结尾是** -5 的连续子数列(**必须包含结尾**)的最大和是多少;
9. **从头开始到结尾是** 4 的连续子数列(**必须包含结尾**)的最大和是多少;

范围缩小了，我们也能从小问题着手了。

解决子问题并找到联系

子问题1 (-2)

显然答案为 -2。
于是子问题1 *(也可以叫作状态1，下文将会都称为状态)* 对应的结果就为 -2。

于是有：

$$result[1] = -2$$

子问题2 $(-2, 1)$

这里要分析与子问题1的联系。 上一个状态 *(也就是上文中的结果)* 满足以下关系：

$$result[1] = -2 <= 0$$

为负数，也就是对数字的增加没有贡献，在本次状态的结果的计算中，我们应该舍弃上一个状态的结果，有：

$$result[2] = a[2] = 1$$

子问题3 $(-2, 1, -3)$

继续分析与上一个子问题的联系。 上一个状态满足以下关系：

$$result[2] = 1 > 0$$

为正数，也就是对数字的增加有贡献，在本次状态的结果的计算中，我们应该保留上一个状态的结果，并加上新增的数字，有：

$$result[3] = result[2] + a[3] = -2$$

注意，下面开始烧脑了。

子问题i (从头开始有i个数字的子数列)

分析与上一个子问题 *(也就是子问题$(i-1)$)* 的联系。

1. 如果上一个状态满足以下关系：

$$result[i-1] > 0$$

对数字的增加有贡献，在本次状态的结果的计算中，我们应该保留上一个状态的结果，并加上新增的数字，有：

$$result[i] = result[i-1] + a[i]$$

2. 如果上一个状态满足以下关系：

$$result[i-1] <= 0$$

对数字的增加没有贡献，在本次状态的结果的计算中，我们应该舍弃上一个状态的结果，有：

$$result[i] = a[i]$$

请试着思考，这样做之后，每个状态储存的到底是啥？ 提示：遍历每个状态，找到状态中最大的值，就是该问题的答案。

对于所有输入的状态转移方程

$$s(n) = max(s(n-1) + a_n, a_n)$$

它可以用C语言写成以下两种形式：

```
//不调用外部函数版
if (s[n-1] > 0) {
    s[n] = s[n-1] + a[n];
}
else {
    s[n] = a[n];
}
```

```
#include <math.h>

s[n] = fmax(s[n-1] + a[n], a[n]);
```

然后找出最大值就好啦！

附代码

这里找出最大值的方法也用到了小技巧。

```
//S(n) = O(n)
//T(n) = O(n)
int GetMaxSumOfSubarray(int *a, int count) {
    //dp => DynamicProgramming
    int *dp = (int *)malloc(sizeof(int) * (count));
    if ((dp) == 0) return -1;

    dp[0] = a[0];
```

```
    int max_result = a[0];

    for (int i = 1; i < count; i++) {
        dp[i] = fmax(dp[i - 1] + a[i], a[i]);
        max_result = fmax(max_result, dp[i]);
    }

    return max_result;
}
```

参考资料·链接

百度百科:无后效性
力扣LeetCode:经典动态规划问题（理解「无后效性」）(liweiwei1419)