0053. 最大子数组和

ITCharge 大约 4 分钟

- 标签：数组、分治、动态规划
- 难度：中等

题目链接

- 0053. 最大子数组和 - 力扣

题目大意

描述：给定一个整数数组 $nums$。

要求：找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

说明：

- **子数组**：指的是数组中的一个连续 。
- $1 \leq nums.length \leq 10^5$。
- $-10^4 \leq nums[i] \leq 10^4$。

示例：

- 示例 1：

```py
输入：nums = [-2,1,-3,4,-1,2,1,-5,4]
输出：6
解释：连续子数组 [4,-1,2,1] 的和最大，为 6。
```

- 示例 2：

```py
输入：nums = [1]
输出：1
```

解题思路

思路 1：动态规划

1. 划分阶段

按照连续子数组的结束位置进行阶段划分。

2. 定义状态

定义状态 $dp[i]$ 为：以第 i 个数结尾的连续子数组的最大和。

3. 状态转移方程

状态 $dp[i]$ 为：以第 i 个数结尾的连续子数组的最大和。则我们可以从「第 $i-1$ 个数结尾的连续子数组的最大和」，以及「第 i 个数的值」来讨论 $dp[i]$。

- 如果 $dp[i-1] < 0$，则「第 $i-1$ 个数结尾的连续子数组的最大和」+「第 i 个数的值」<「第 i 个数的值」，即：$dp[i-1] + nums[i] < nums[i]$。所以，此时 $dp[i]$ 应取「第 i 个数的值」，即 $dp[i] = nums[i]$。
- 如果 $dp[i-1] \geq 0$，则「第 $i-1$ 个数结尾的连续子数组的最大和」 +「第 i 个数的值」>= 第 i 个数的值，即：$dp[i-1] + nums[i] \geq nums[i]$。所以，此时 $dp[i]$ 应取「第 $i-1$ 个数结尾的连续子数组的最大和」+「 第 i 个数的值」，即 $dp[i] = dp[i-1] + nums[i]$。

归纳一下，状态转移方程为：

$$dp[i] = \begin{cases} nums[i], & dp[i-1] < 0 \\ dp[i-1] + nums[i] & dp[i-1] \geq 0 \end{cases}$$

4. 初始条件

- 第 0 个数结尾的连续子数组的最大和为 $nums[0]$，即 $dp[0] = nums[0]$。

5. 最终结果

根据状态定义，$dp[i]$ 为：以第 i 个数结尾的连续子数组的最大和。则最终结果应为所有 $dp[i]$ 的最大值，即 $max(dp)$。

思路 1：代码

```py
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        size = len(nums)
        dp = [0 for _ in range(size)]

        dp[0] = nums[0]
        for i in range(1, size):
            if dp[i - 1] < 0:
                dp[i] = nums[i]
            else:
                dp[i] = dp[i - 1] + nums[i]
        return max(dp)
```

思路 1：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 为数组 $nums$ 的元素个数。
- **空间复杂度**：$O(n)$。

思路 2：动态规划 + 滚动优化

因为 $dp[i]$ 只和 $dp[i-1]$ 和当前元素 $nums[i]$ 相关，我们也可以使用一个变量 $subMax$ 来表示以第 i 个数结尾的连续子数组的最大和。然后使用 $ansMax$ 来保存全局中最大值。

思路 2：代码

```py
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        size = len(nums)
        subMax = nums[0]
        ansMax = nums[0]

        for i in range(1, size):
```

```
            if subMax < 0:
                subMax = nums[i]
            else:
                subMax += nums[i]
            ansMax = max(ansMax, subMax)
        return ansMax
```

思路 2：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 为数组 $nums$ 的元素个数。
- **空间复杂度**：$O(1)$。

思路 3：分治算法

我们将数组 $nums$ 根据中心位置分为左右两个子数组。则具有最大和的连续子数组可能存在以下 3 种情况：

1. 具有最大和的连续子数组在左子数组中。
2. 具有最大和的连续子数组在右子数组中。
3. 具有最大和的连续子数组跨过中心位置，一部分在左子数组中，另一部分在右子树组中。

那么我们要求出具有最大和的连续子数组的最大和，则分别对上面 3 种情况求解即可。具体步骤如下：

1. 将数组 $nums$ 根据中心位置递归分为左右两个子数组，直到所有子数组长度为 1。
2. 长度为 1 的子数组最大和肯定是数组中唯一的数，将其返回即可。
3. 求出左子数组的最大和 $leftMax$。
4. 求出右子树组的最大和 $rightMax$。
5. 求出跨过中心位置，一部分在左子数组中，另一部分在右子树组的子数组最大和 $leftTotal + rightTotal$。
6. 求出 3、4、5 中的最大值，即为当前数组的最大和，将其返回即可。

思路 3：代码

```py
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        def max_sub_array(low, high):
            if low == high:
                return nums[low]

            mid = low + (high - low) // 2
            leftMax = max_sub_array(low, mid)
            rightMax = max_sub_array(mid + 1, high)

            total = 0
            leftTotal = -inf
            for i in range(mid, low - 1, -1):
                total += nums[i]
                leftTotal = max(leftTotal, total)

            total = 0
            rightTotal = -inf
            for i in range(mid + 1, high + 1):
                total += nums[i]
                rightTotal = max(rightTotal, total)

            return max(leftMax, rightMax, leftTotal + rightTotal)

        return max_sub_array(0, len(nums) - 1)
```

思路 3：复杂度分析

- **时间复杂度**：$O(n)$。
- **空间复杂度**：$O(\log n)$。