

# 最大和连续子数组

## 题述

### 53. 最大子数组和

难度 **简单**

4706



给你一个整数数组 `nums`，请你找出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。

**子数组** 是数组中的一个连续部分。

示例 1:

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:

输入: `nums = [1]`

输出: 1

示例 3:

输入: `nums = [5,4,-1,7,8]`

输出: 23

## 思路

### 暴力

双重for循环

- 第一层for设置起始位置
- 第二层for循环遍历数组寻找最大值

### 动态规划

#### 1、确定dp数组（dp table）以及下标的含义

`dp[i]`: 包括下标*i*之前的最大连续子序列和为`dp[i]`。

## 2、确定递推公式

$dp[i]$ 只有两个方向可以推出来：

- $dp[i - 1] + nums[i]$ ，即： $nums[i]$ 加入当前连续子序列和
- $nums[i]$ ，即：从头开始计算当前连续子序列和

## 3、dp数组如何初始化

从递推公式可以看出来 $dp[i]$ 是依赖于 $dp[i - 1]$ 的状态， $dp[0]$ 就是递推公式的基础。

根据 $dp[i]$ 的定义，很明显 $dp[0]$ 应为 $nums[0]$ 即 $dp[0] = nums[0]$ 。

## 4、确定遍历顺序

递推公式中 $dp[i]$ 依赖于 $dp[i - 1]$ 的状态，需要从前向后遍历。

## 5、举例推导dp数组

以示例一为例，输入： $nums = [-2,1,-3,4,-1,2,1,-5,4]$ ，对应的dp状态如下：

输入: [-2,1,-3,4,-1,2,1,-5,4]

下标: 0      1      2      3      4      5      6      7      8

dp[i]:

-2	1	-2	4	3	5	6	1	5
----	---	----	---	---	---	---	---	---

## 分治

分治法的思想类似于快速排序算法

将输入数组划分为左右两部分，我们要求的最大子序和要么在左半边，要么在右半边，要么就是穿过中间，我们可以用递归进行处理。

## 题解

### Python

### 暴力

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        #暴力解法
        result = -float('inf') # -float('inf')为负无穷 相当于C++中的INT_MIN
        count = 0
        for i in range(len(nums)):
            count = 0
            j = i
            for j in range(j, len(nums)):
                count += nums[j]
                if count > result:
                    result = count
            else:
```

```
        result = result
    return result
```

```
from typing import List
def maxSubArray(nums: List[int]) -> int:
    #暴力解法
    result = -float('inf')    # -float('inf')为负无穷 相当于C++中的INT_MIN
    count = 0
    start = 0
    end = 0
    for i in range(len(nums)):
        count = 0
        j = i
        for j in range(j, len(nums)):
            count += nums[j]
            if count > result:
                result = count
                start = i
                end = j
            else:
                result = result
    return result, start, end
```

## 分治

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        # 分治算法
        n = len(nums)    # 数组长度
        if n == 1:
            return nums[0]
        else:
            #递归计算左半边的最大子数组和
            MaxSubLeft = self.maxSubArray(nums[0: len(nums) // 2])
            #递归计算右半边的最大子数组和
            MaxSubRight = self.maxSubArray(nums[len(nums) // 2: len(nums)])

            # 计算中间的最大子数组和 从右至左计算左边的最大子数组和 从左至右计算右边的最大子数组和
            Max_Left = nums[len(nums) // 2 - 1]
            temp = 0
            for i in range(len(nums) // 2 - 1, -1, -1):
                temp += nums[i]
                Max_Left = max(temp, Max_Left)

            Max_Right = nums[len(nums) // 2]
            temp = 0
            for i in range(len(nums) // 2, len(nums)):
                temp += nums[i]
                Max_Right = max(temp, Max_Right)

            #返回最终结果
            return max(MaxSubLeft, MaxSubRight, Max_Left+Max_Right)
```

## 动态规划

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        # 动态规划
        if len(nums) == 0:
            return 0

        #初始化dp数组
        dp = [0] * len(nums)
        dp[0] = nums[0]
        result = dp[0]
        for i in range(1,len(nums)):
            dp[i] = max(dp[i-1] + nums[i], nums[i])    #递推公式
            result = max(result,dp[i]) #result负责保存dp[i]中的最大值
        return result
```

## 思考

---