算法-动态规划

动态规划方法概述

动态规划中每一个状态一定是由上一个状态推导出来的（想一下上课时老师所举的路径由后往前推得的例子），这是它区别于贪心算法的关键点。

> *动态规划是由前一个状态依次推导而得*

动态规划一般解题步骤

做DP的题目时，时常会陷入一个误区：那就是将状态转移公式（递推公式)死运用，而没有进行实际模拟。

状态转移公式（递推公式）是很重要，但动规不仅仅只有递推公式。

动态规划问题可以拆解为以下五个步骤，咱们在实际题目中逐渐融会贯通：

- 确定dp数组（dp table 也就是上课时说的辅助备忘录)以及其下标的含义
- 确定状态转移方程（递推公式）
- dp数组应该如何初始化?
- 确定遍历顺序
- 举例模拟推导dp数组

典例1-斐波那契数列

509. 斐波那契数

难度 简单 435

斐波那契数 （通常用 `F(n)` 表示）形成的序列称为 **斐波那契数列** 。该数列由 `0` 和 `1` 开始，后面的每一项数字都是前面两项数字的和。也就是：

```
F(0) = 0，F(1) = 1
F(n) = F(n - 1) + F(n - 2)，其中 n > 1
```

给定 `n` ，请计算 `F(n)` 。

示例 1：

```
输入：n = 2
输出：1
解释：F(2) = F(1) + F(0) = 1 + 0 = 1
```

示例 2：

```
输入：n = 3
输出：2
解释：F(3) = F(2) + F(1) = 1 + 1 = 2
```

示例 3：

```
输入：n = 4
输出：3
解释：F(4) = F(3) + F(2) = 2 + 1 = 3
```

```python
class Solution:
    def fib(self, n: int) -> int:
        # 动态规划
        if n<= 1:
            return n
        dp=[0] * (n+1) #定义dp数组 长度为n+1 全部初始化为0
        dp[0] = 0
        dp[1] = 1
        for i in range(2,n+1):
            dp[i] = dp[i-1] + dp[i-2]    #递推公式
        return dp[n]
```

实验2-最大和连续子数组

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
```

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。

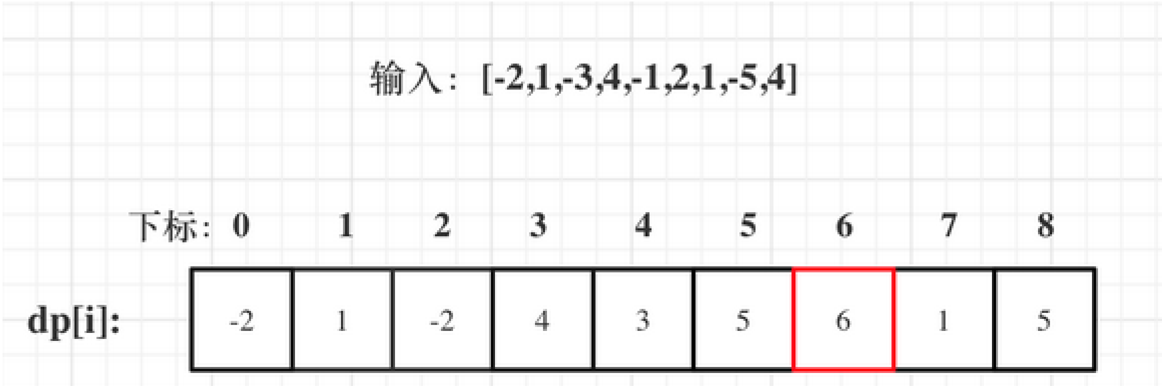
根据dp[i]的定义，很明显dp[0]应为nums[0]即dp[0] = nums[0]。

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：




```python
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
```

实验2-数组等差子序列

446. 等差数列划分 II - 子序列

难度 困难 253 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `nums` ，返回 `nums` 中所有 **等差子序列** 的数目。

如果一个序列中 **至少有三个元素** ，并且任意两个相邻元素之差相同，则称该序列为等差序列。

- 例如，`[1, 3, 5, 7, 9]`、`[7, 7, 7, 7]` 和 `[3, -1, -5, -9]` 都是等差序列。
- 再例如，`[1, 1, 2, 5, 7]` 不是等差序列。

数组中的子序列是从数组中删除一些元素（也可能不删除）得到的一个序列。

- 例如，`[2,5,10]` 是 `[1,2,1,2,4,1,5,10]` 的一个子序列。

题目数据保证答案是一个 **32-bit** 整数。

示例 1：

```
输入：nums = [2,4,6,8,10]
输出：7
解释：所有的等差子序列为：
[2,4,6]
[4,6,8]
[6,8,10]
[2,4,6,8]
[4,6,8,10]
[2,4,6,8,10]
[2,6,10]
```

示例 2：

```
输入：nums = [7,7,7,7,7]
输出：16
解释：数组中的任意子序列都是等差子序列。
```

本题解是基于常规的思考思路，没有任何奇思妙解。
首先我们从题目入手，子序列问题一般可以考虑用动态规划来解决，因此我们从这个方面来思考：

动态规划的状态设计为

```
dp[i][j]：以nums[i]为结尾的子序列，前一个等差数字是nums[j]。
```

动态规划的转移方程设计为

```
dp[i][j] += dp[j][k] + 1，其中nums[k]是在nums[j]之前的等差数字。
```

在此过程中，怎么快速找到在nums[j]之前的等差数字nums[k]呢？可以采用哈希表来预存储所有列表中的数字以及对应的索引！
接下来就很简单了，直接看代码好了。

```
class Solution:
    def numberOfArithmeticSlices(self, nums: List[int]) -> int:
        if len(nums) < 3:
            return 0

        #记录数字以及对应的所有索引
```

```python
        index_dict = collections.defaultdict(list)
        for i in range(len(nums)):
            index_dict[nums[i]].append(i)

        #动态规划方程
        dp = [[0 for _ in range(len(nums))] for _ in range(len(nums))]

        res = 0
        for i in range(len(nums)):
            for j in range(i):
                #寻找nums[k]
                tar = 2 * nums[j] - nums[i]
                #nums[k]必须出现过
                if tar in index_dict:
                    tar_index = index_dict[tar]
                    for k in range(len(tar_index)):
                        #k必须在j之前
                        if tar_index[k] >= j:
                            break
                        dp[i][j] += dp[j][tar_index[k]] + 1
                res += dp[i][j]
        return res
```