

## 动态规划

### 斐波那契数列

$$f(n+1) = f(n) + f(n-1) \quad f(n+1) = f(n) + f(n-1)$$

```
class Solution {
    // time O(n) space O(1)
public:
    int mod = 1e9+7;
    int fib(int n) {
        int a=0,b=1,sum;
        for (int i = 0; i < n; ++i) {
            sum=(a+b) % mod;
            a=b;
            b=sum;
        }
        return a;
    }
};
```

### 连续子数组的最大和

nums

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

dp

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

状态定义：

$dp[i]$  代表以元素  $nums[i]$  为结尾的连续子数组最大和

转移方程：

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

```

xxxxxxxxx class Solution {public:    int maxSubArray(vector<int>&
nums) {        // dp        int res = nums[0];        for(int i = 1; i < nums.size();
i++) {            nums[i] += max(nums[i - 1], 0);            res = max(res, nums[i]);
        }        return res;    }};

```

## 剑指 Offer 47. 礼物的最大价值

在一个  $m \times n$  的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物？

示例 1:

输入:

```

[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]

```

输出: 12

解释: 路径 1→3→5→2→1 可以拿到最多价值的礼物

*grid*

1	3	1
1	5	1
4	2	1

*dp*

1	4	5
2	9	10
6	11	12

例如图中:  $f(1,1) = \max[f(0,1), f(1,0)] + \text{grid}(1,1)$

设  $f(i,j)$  代表从棋盘的左上角开始到达单元格  $(i,j)$  时能拿到礼物的最大累计价值。则易得到以下递推关系:

$$f(i,j) = \max[f(i-1,j), f(i,j-1)] + \text{grid}(i,j)$$

动态规划解析：

- **状态定义：** 设动态规划矩阵  $dp$  ,  $dp(i, j)$  代表从棋盘的左上角开始，到达单元格  $(i, j)$  时能拿到礼物的最大累计价值。
- **转移方程：**
  1. 当  $i=0$  且  $j=0$  时，为起始元素；
  2. 当  $i=0$  且  $j \neq 0$  时，为矩阵第一行元素，只可从左边到达；
  3. 当  $i \neq 0$  且  $j=0$  时，为矩阵第一列元素，只可从上边到达；
  4. 当  $i \neq 0$  且  $j \neq 0$  时，可从左边或上边到达；

$$dp(i, j) = \begin{cases} grid(i, j) & , i=0, j=0 \\ grid(i, j) + dp(i, j-1) & , i=0, j \neq 0 \\ grid(i, j) + dp(i-1, j) & , i \neq 0, j=0 \\ grid(i, j) + \max[dp(i-1, j), dp(i, j-1)] & , i \neq 0, j \neq 0 \end{cases}$$

- **初始状态：**  $dp[0][0] = grid[0][0]$  , 即到达单元格  $(0, 0)$  时能拿到礼物的最大累计价值为  $grid[0][0]$  ；
- **返回值：**  $dp[m-1][n-1]$  ,  $m, n$  分别为矩阵的行高和列宽，即返回  $dp$  矩阵右下角元素。

空间复杂度优化：

- 由于  $dp[i][j]$  只与  $dp[i-1][j]$  ,  $dp[i][j-1]$  ,  $grid[i][j]$  有关系，因此可以将原矩阵  $grid$  用作  $dp$  矩阵，即直接在  $grid$  上修改即可。
- 应用此方法可省去  $dp$  矩阵使用的额外空间，因此空间复杂度从  $O(MN)$  降至  $O(1)$  。

```
class Solution {
public:
    int maxValue(vector<vector<int>>& grid) {
        // f(i, j) 为从棋盘左上角走至单元格 (i, j) 的礼物最大累计价值
        int m = grid.size(), n = grid[0].size();
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                if (i == 0 && j == 0) continue;
                if (i == 0) {
                    grid[i][j] += grid[i][j - 1]; // dp
                }
                else if (j == 0) {
                    grid[i][j] += grid[i - 1][j];
                }
                else {
                    grid[i][j] += max(grid[i - 1][j], grid[i][j - 1]);
                }
            }
        }
        return grid[m - 1][n - 1];
    }
};
```

## 剑指 Offer 46. 把数字翻译成字符串

给定一个数字，我们按照如下规则把它翻译为字符串：0 翻译成“a”，1 翻译成“b”，……，11 翻译成“l”，……，25 翻译成“z”。一个数字可能有多个翻译。请编程实现一个函数，用来计算一个数字有多少种不同的翻译方法。

示例 1:

输入: 12258

输出: 5

解释: 12258有5种不同的翻译，分别是"bccfi", "bwfi", "bczi", "mcfi"和"mzi"

$num = x_1x_2 \dots x_{i-2}x_{i-1}x_i \dots x_{n-1}x_n$

(例如:  $12258 = x_1x_2x_3x_4x_5$ )



设  $x_1x_2 \dots x_{i-2}$  的翻译方案数量为  $f(i-2)$   
设  $x_1x_2 \dots x_{i-2}x_{i-1}$  的翻译方案数量为  $f(i-1)$



当整体翻译  $x_{i-1}x_i$  时,  $x_1x_2 \dots x_{i-2}x_{i-1}x_i$  的方案数为  $f(i-2)$   
当单独翻译  $x_i$  时,  $x_1x_2 \dots x_{i-2}x_{i-1}x_i$  的方案数为  $f(i-1)$



方案数的递推关系:

$$f(i) = \begin{cases} f(i-2) + f(i-1) & \text{若数字 } x_{i-1}x_i \text{ 可被翻译} \\ f(i-1) & \text{若数字 } x_{i-1}x_i \text{ 不可被翻译} \end{cases}$$

```
class Solution {
public:
    int translateNum(int num) {
        string stringNum = to_string(num);
        int n = stringNum.size();
        int dp[n];
        int a = 1, b = 1, c; // a=dp[i-1] b=dp[i-2] b ----> a ----> c

        for (int i = 2; i <= n; i++) {
            string tmp = stringNum.substr(i - 2, 2);
            if (tmp >= "10" && tmp <= "25") {
                c = a + b;
            } else {
                c = a;
            }
        }
    }
}
```

```
        b = a;  
        a = c;  
    }  
    return a;  
}  
};
```