【第二十五周】递推算法及解题套路

1、256. 粉刷房子

假如有一排房子，共 n 个，每个房子可以被粉刷成红色、蓝色或者绿色这三种颜色中 的一种，你需要粉刷所有的房子并且使其相邻的两个房子颜色不能相同。

当然，因为市场上不同颜色油漆的价格不同，所以房子粉刷成不同颜色的花费成本也 是不同的。每个房子粉刷成不同颜色的花费是以一个 n x 3 的正整数矩阵 costs 来表 示的。

例如， costs[0][0] 表示第 0 号房子粉刷成红色的成本花费； costs[1][2] 表示第 1 号 房子粉刷成绿色的花费，以此类推。 请计算出粉刷完所有房子最少的花费成本。

示例：

```
输入:costs = [[17,2,17],[16,16,5],[14,3,19]]
输出:10
解释:将 0 号房子粉刷成蓝色，1 号房子粉刷成绿色，2 号房子粉刷成蓝色。
最少花费: 2 + 5 + 3 = 10。
```

```
 /**
 * @param {number[][]} costs
 * @return {number}
 */
// 思路：dp[i][j]表示当前第i个房子涂的第j个颜色，当前的价格为之前房子dp[i-1][x1]、dp[i-1][x2]最小值（x1，x2为3个颜色除去颜色j后剩下的另外两个颜色）的价格加上当前costs[i][j]的价格，最后比较dp末尾三个元素，取最小值。
var minCost = function(costs) {
    let n = costs.length;
    let dp = new  Array(n);
    for (let i = 0; i < dp.length; i++) dp[i] = new Array(3);
    dp[0][0] = costs[0][0], dp[0][1] = costs[0][1], dp[0][2] = costs[0][2];
    for (let i = 1; i < n; i++){
        dp[i][0] = Math.min(dp[i-1][1], dp[i-1][2]) + costs[i][0];
        dp[i][1] = Math.min(dp[i-1][0], dp[i-1][2]) + costs[i][1];
        dp[i][2] = Math.min(dp[i-1][1], dp[i-1][0]) + costs[i][2];
    }
    return Math.min(Math.min(dp[dp.length - 1][0], dp[dp.length - 1][1]),
dp[dp.length - 1][2]);
};
```

2、119. 杨辉三角 II

1、外层循环从上往下一层层求，复用一维数组 res（滚动数组），计算每个位置上的元素，只取决于上一行的值。

2、内层遍历的递推式为`res[j] = res[j] + res[j-1]`，要保证等号右边的两个，是上一行的“旧值”。

3、如果内层遍历是从左往右，会出现`res[j-1]`是本行的上一轮迭代求出的新值，不是上一行的旧值。

4、如果内层遍历是从左往右，会出现`res[j-1]`是本行的上一轮迭代求出的新值，不是上一行的旧值。

```
var getRow = function(rowIndex) {
    const res = new Array(rowIndex + 1);
    res[0] = 1;

    for (let i = 1; i < rowIndex + 1; i++) {
        res[0] = res[i] = 1;
        for (let j = i - 1; j >= 1; j--) {
            res[j] = res[j] + res[j - 1];
        }
    }
    return res;
};
```

3、152. 乘积最大子数组

1、计算子数组乘积公式：dp[n] = Math.max((dp[n-1] * val[n],val[n])

2、考虑到有负数的情况，需要记录一个最大值，一个最小值，在有负数的情况下，最大值乘以最小值变成最小值，最小值乘以负数就是最大。又因为dp[n]的大小只和dp[n-1]有关系。所以，只需要记录前一个最大值和最小值。直接用变量处理

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var maxProduct = function(nums) {
    // ans:当前找到的连续子数组的最大值，max_num：前一个最大值，min_num：前一个最小值，因为是乘法关系，所以初始化为1
    let ans = -Infinity,max_num = 1,min_num = 1;
    for(const x of nums){
        // 大小关系发生颠倒
        if(x < 0){
            let temp;
            temp = max_num;
            max_num = min_num;
            min_num = temp;
        }
        max_num = Math.max(x * max_num,x);
        min_num = Math.min(x * min_num,x);
        ans = Math.max(ans,max_num);
    }
    return ans;
};
```

4、198. 打家劫舍

1、二维数组 dp[i][x，y]。i：每一间屋子 x：当前屋子不偷的金额 y：当前屋子偷的金额。

2、0: 不偷 1: 偷 第一间屋子偷 或者 不偷 赋值

3、当前屋子不偷，那上一间屋子可以偷，也可以不偷，所以取上间屋子两者最大值

4、当前屋子偷，那上一间屋子就不能偷，不偷的值 + 当前屋子的值

5、取 最后一项偷或者不偷的最大值

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var rob = function(nums) {
    let n = nums.length;
    // 状态数组
    // i：每一间屋子    x：当前屋子不偷的金额     y：当前屋子偷的金额
    // 二维数组 dp[i][x，y]
    let dp =  new Array(n).fill(0).map(item => item = new Array(2));
    // 0: 不偷 1: 偷  第一间屋子偷 或者 不偷 赋值
    dp[0][0] = 0, dp[0][1] = nums[0];
    for(let i = 1; i < n;i++){
        // 当前屋子不偷，那上一间屋子可以偷，也可以不偷，所以取上间屋子两者最大值
        dp[i][0] = Math.max(dp[i-1][0],dp[i-1][1]);
        // 当前屋子偷，那上一间屋子就不能偷，不偷的值 + 当前屋子的值
        dp[i][1] = dp[i - 1][0] + nums[i];
    }
     // 取 最后一项偷或者不偷的最大值
    return Math.max(dp[n-1][0],dp[n-1][1]);
};
```

5、122. 买卖股票的最佳时机 II

1、让股票后一天的价格 减去 前一个的价格 ，如果是正数，证明股票正在收益；否则股票在赔钱。

2、所以就是相邻两项做差，把所有差值大于0的进行相加。

```
/**
 * @param {number[]} prices
 * @return {number}
 */
var maxProfit = function(prices) {
    let ans = 0;
    for(let i = 1;i < prices.length;i++){
        if(prices[i] > prices[i - 1]) ans += prices[i] - prices[i-1];
    }
    return ans;
};
```

开课吧