【第二十五周】递推算法及解题套路

70. 爬楼梯

假设你正在爬楼梯。需要 *n* 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

注意：给定 *n* 是一个正整数。

示例 ：

```
输入： 2
输出： 2
解释： 有两种方法可以爬到楼顶。
1.  1 阶 + 1 阶
2.  2 阶
```

```
class Solution {
public:
    int climbStairs(int n) {
        vector<int> f(n + 1);
        f[0] = 1, f[1] = 1;
        for (int i = 2; i <= n; i++) f[i] = f[i - 1] + f[i - 2];
        return f[n];
    }
};
```

746. 使用最小花费爬楼梯

数组的每个下标作为一个阶梯，第 `i` 个阶梯对应着一个非负数的体力花费值 `cost[i]`（下标从 `0` 开始）。

每当你爬上一个阶梯你都要花费对应的体力值，一旦支付了相应的体力值，你就可以选择向上爬一个阶梯或者爬两个阶梯。

请你找出达到楼层顶部的最低花费。在开始时，你可以选择从下标为 0 或 1 的元素作为初始阶梯。

示例 ：

```
输入：cost = [10, 15, 20]
输出：15
```

```
解释：最低花费是从 cost[1] 开始，然后走两步即可到阶梯顶，一共花费 15 。
```

```
class Solution {
public:
    int minCostClimbingStairs(vector<int>& cost) {
        int n = cost.size();
        vector<int> dp(n + 1);
        cost.push_back(0);
        dp[0] = cost[0]; dp[1] = cost[1];
        for (int i = 2; i <= n; i++) dp[i] = min(dp[i - 1], dp[i - 2]) + cost[i];
        return dp[n];
    }
};
```

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。
```

```
class Solution {
public:
    int minCost(vector<vector<int>>& costs) {
        int n = costs.size();
        vector<vector<int>> dp;
        for (int i = 0; i < 2; i++) dp.push_back(vector<int>(3));
        for (int i = 0; i < 3; i++) dp[0][i] = costs[0][i];
        for (int i = 1; i < n; i++) {
            int ind = i % 2;
            int pre_ind = !ind;
            dp[ind][0] = min(dp[pre_ind][1], dp[pre_ind][2]) + costs[i][0];
            dp[ind][1] = min(dp[pre_ind][0], dp[pre_ind][2]) + costs[i][1];
            dp[ind][2] = min(dp[pre_ind][0], dp[pre_ind][1]) + costs[i][2];
```

```
        }
        int ind = (n - 1) % 2;
        return min(dp[ind][0], min(dp[ind][1], dp[ind][2]));
    }
};
```

120. 三角形最小路径和

给定一个三角形 `triangle` ，找出自顶向下的最小路径和。

每一步只能移动到下一行中相邻的结点上。**相邻的结点** 在这里指的是 **下标** 与 **上一层结点下标** 相同或者等于 **上一层结点下标 + 1** 的两个结点。也就是说，如果正位于当前行的下标 `i` ，那么下一步可以移动到下一行的下标 `i` 或 `i + 1` 。

示例 ：

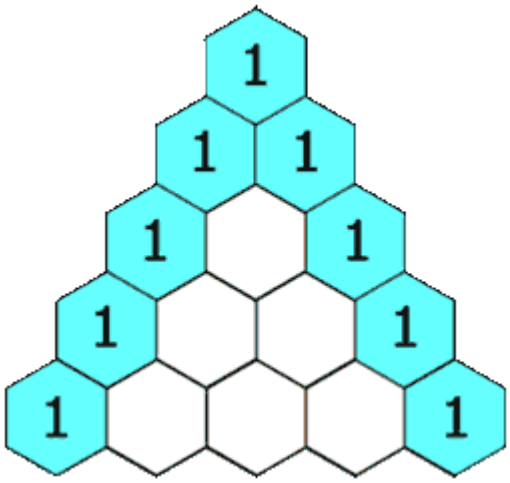
```
输入：triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]
输出：11
解释：如下面简图所示：
2
3 4
6 5 7
4 1 8 3
自顶向下的最小路径和为 11（即，2 + 3 + 5 + 1 = 11）。
```

```
class Solution {
public:
    int minimumTotal(vector<vector<int>>& triangle) {
        int n = triangle.size();
        vector<vector<int>> dp;
        for (int i = 0; i < 2; i++) dp.push_back(vector<int>(n));
        for (int i = 0; i < n; i++) dp[(n - 1) % 2][i] = triangle[n - 1][i];
        for (int i = n - 2; i >= 0; --i) {
            int ind = i % 2;
            int next_ind = !ind;
            for (int j = 0; j <= i; j++) {
                dp[ind][j] = min(dp[next_ind][j], dp[next_ind][j + 1]) + triangle[i]
[j];
            }
        }
        return dp[0][0];
    }
};
```

119. 杨辉三角 II

给定一个非负索引 `rowIndex`，返回「杨辉三角」的第 `rowIndex` 行。

在「杨辉三角」中，每个数是它左上方和右上方的数的和。



示例：

```
输入: rowIndex = 3
输出: [1,3,3,1]
```

```
class Solution {
public:
    vector<int> getRow(int rowIndex) {
        int n = rowIndex + 1;
        vector<vector<int>> f;
        for (int i = 0; i < 2; i++) f.push_back(vector<int>(n));
        for (int i = 0; i < n; i++) {
            int ind = i % 2;
            int pre_ind = !ind;
            f[ind][0] = 1;
            for (int j = 1; j <= i; j++) {
                f[ind][j] = f[pre_ind][j] + f[pre_ind][j - 1];
            }
        }
        return f[(n - 1) % 2];
    }
};
```

53. 最大子序和

给定一个整数数组 `nums` ，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。

示例 ：

```
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 。
```

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        for (int i = 1; i < nums.size(); i++) nums[i] += nums[i - 1];
        int pre = 0, ans = INT_MIN;
        for (auto x : nums) {
            ans = max(x - pre, ans);
            pre = min(x, pre);
        }
        return ans;
    }
};
```

122. 买卖股票的最佳时机 II

给定一个数组 `prices` ，其中 `prices[i]` 是一支给定股票第 `i` 天的价格。

设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

示例：

```
输入: prices = [7,1,5,3,6,4]
输出: 7
解释: 在第 2 天（股票价格 = 1）的时候买入，在第 3 天（股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5-1 = 4 。
     随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，这笔交易所能获得利润 = 6-3 = 3 。
```

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int ans = 0;
        for (int i = 1; i < prices.size(); i++) {
            if (prices[i] > prices[i - 1]) ans += prices[i] - prices[i - 1];
        }
        return ans;
    }
};
```

198. 打家劫舍

你是一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，**如果两间相邻的房屋在**

同一晚上被小偷闯入，系统会自动报警。

给定一个代表每个房屋存放金额的非负整数数组，计算你 **不触动警报装置的情况下** ，一夜之内能够偷窃到的最高金额。

示例 ：

```
输入：[1,2,3,1]
输出：4
解释：偷窃 1 号房屋 (金额 = 1) ，然后偷窃 3 号房屋 (金额 = 3)。
     偷窃到的最高金额 = 1 + 3 = 4 。
```

```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
        vector<vector<int>> dp;
        for (int i = 0; i < n; i++) dp.push_back(vector<int>(2));
        dp[0][0] = 0, dp[0][1] = nums[0];
        for (int i = 1; i < n; i++) {
            dp[i][0] = max(dp[i - 1][0], dp[i - 1][1]);
            dp[i][1] = dp[i - 1][0] + nums[i];
        }
        return max(dp[n - 1][0], dp[n - 1][1]);
    }
};
```

152. 乘积最大子数组

给你一个整数数组 `nums` ，请你找出数组中乘积最大的连续子数组（该子数组中至少包含一个数字），并返回该子数组所对应的乘积。

示例 1:

```
输入: [2,3,-2,4]
输出:6解释: 子数组 [2,3] 有最大乘积 6。
```

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int ans = INT_MIN, max_num = 1, min_num = 1;
        for (auto x : nums) {
            if (x < 0) swap(max_num, min_num);
            max_num = max(x * max_num, x);
            min_num = min(x * min_num, x);
            ans = max(ans, max_num);
        }
```

```
        return ans;
    }
};
```

322. 零钱兑换

给你一个整数数组 `coins` ，表示不同面额的硬币；以及一个整数 `amount` ，表示总金额。

计算并返回可以凑成总金额所需的 **最少的硬币个数** 。如果没有任何一种硬币组合能组成总金额，返回 `-1` 。

你可以认为每种硬币的数量是无限的。

示例 1：

```
输入：coins =[1, 2, 5], amount =11输出：3解释：11 = 5 + 5 + 1
```

```
class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        vector<int> dp(amount + 1);
        dp[0] = 0;
        for (int i = 1; i <= amount; i++) dp[i] = -1;
        for (int i = 1; i <= amount; i++) {
            for (auto x : coins) {
                if (i < x) continue;
                   if (dp[i - x] == -1) continue;
                if (dp[i] == -1 || dp[i] > dp[i - x] + 1) dp[i] = dp[i - x] + 1;
            }
        }
        return dp[amount];
    }
};
```

300. 最长递增子序列

给你一个整数数组 `nums` ，找到其中最长严格递增子序列的长度。

子序列是由数组派生而来的序列，删除（或不删除）数组中的元素而不改变其余元素的顺序。例如，`[3,6,2,7]` 是数组 `[0,3,1,6,2,2,7]` 的子序列。

示例 1：

```
输入：nums = [10,9,2,5,3,7,101,18]
输出：4
解释：最长递增子序列是 [2,3,7,101]，因此长度为 4 。
```

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        vector<int> dp(nums.size());
        int ans = 0;
        for (int i = 0; i < nums.size(); i++) {
            dp[i] = 1;
            for (int j = 0; j < i; j++) {
                if (nums[j] >= nums[i]) continue;
                dp[i] = max(dp[i], dp[j] + 1);
            }
            ans = max(dp[i], ans);
        }
        return ans;
    }
};
```