动态规划递推公式速查

1.斐波那契问题、爬楼梯问题

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

dp数组的含义：**第i月份的兔子数量、走到第i层的不同的方法**

递推公式：**dp[i]=dp[i-1]+dp[i-2]**

2.爬楼梯的最小花费

```
输入：cost = [1,100,1,1,1,100,1,1,100,1]
输出：6
解释：你将从下标为 0 的台阶开始。
- 支付 1 ，向上爬两个台阶，到达下标为 2 的台阶。
- 支付 1 ，向上爬两个台阶，到达下标为 4 的台阶。
- 支付 1 ，向上爬两个台阶，到达下标为 6 的台阶。
- 支付 1 ，向上爬一个台阶，到达下标为 7 的台阶。
- 支付 1 ，向上爬两个台阶，到达下标为 9 的台阶。
- 支付 1 ，向上爬一个台阶，到达楼梯顶部。
总花费为 6 。
```

dp数组的含义：**跳到第i层的最小花费**

递推公式为：**dp[i]=min(dp[i-1]+cost[i-1],dp[i-2]+cost[i-2])**

3.不同路径

```
输入：m = 3, n = 2
输出：3
解释：
从左上角开始，总共有 3 条路径可以到达右下角。
1. 向右 -> 向下 -> 向下
2. 向下 -> 向下 -> 向右
3. 向下 -> 向右 -> 向下
```

dp数组的含义：**移动到(i,j)下标的位置时的不同路径为dp[i][j]**

递推公式：**dp[i][j]=dp[i-1][j]+dp[i][j-1]**

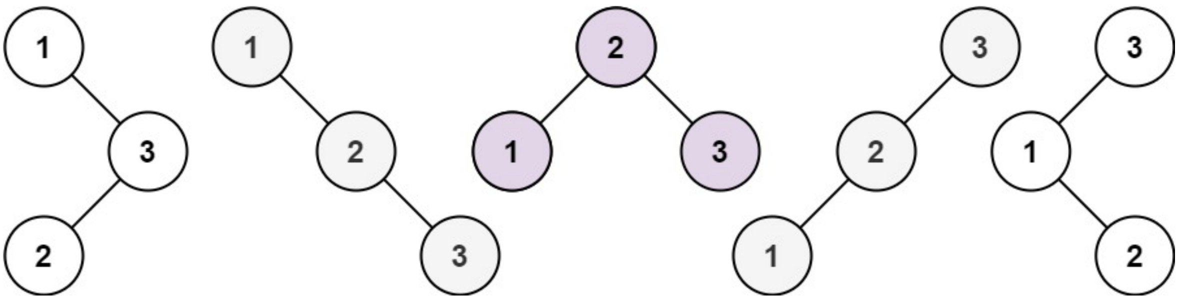
4.整数拆分

```
输入: n = 10
输出: 36
解释: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36。
```

dp数组的含义：**下标为i的数，拆分成若干个数相乘后的最大值**

递推公式：**dp[i] = max(j * (i - j), j * dp[i-j],dp[i]);**

5.不同的二叉搜索树



dp数组的含义：**有i个节点的不同的二叉搜索树的种类为dp[i]**

递推公式为：**dp[i]+=dp[j]*dp[i-1-j]**

6.01背包问题

dp数组的含义：**容量为j的背包的能装的最大价值为dp[j]**

递推公式：**dp[j]=max(dp[j],dp[j-weight[i]]+values[i])**

7.分割等和子集

```
输入：nums = [1,5,11,5]
输出：true
```

```
解释：数组可以分割成 [1, 5, 5] 和 [11] 。
```

dp数组的含义是：**容量为j的背包所能放的最大物品的总重量为dp[j]**

递推公式为：**dp[j]=max(dp[j],dp[j-nums[i]]+nums[i])**

8.最后一块石头的重量

```
输入：stones = [2,7,4,1,8,1]
输出：1
解释：
组合 2 和 4，得到 2，所以数组转化为 [2,7,1,8,1]，
组合 7 和 8，得到 1，所以数组转化为 [2,1,1,1]，
组合 2 和 1，得到 1，所以数组转化为 [1,1,1]，
组合 1 和 1，得到 0，所以数组转化为 [1]，这就是最优值。
```

dp数组的含义是：**容量为j的背包所能放的最大物品的总重量为dp[j]**

递推公式为：**dp[j]=max(dp[j],dp[j-nums[i]]+nums[i])**

9.目标和

```
输入：nums = [1,1,1,1,1], target = 3
输出：5
解释：一共有 5 种方法让最终目标和为 3 。
-1 + 1 + 1 + 1 + 1 = 3
+1 - 1 + 1 + 1 + 1 = 3
+1 + 1 - 1 + 1 + 1 = 3
+1 + 1 + 1 - 1 + 1 = 3
+1 + 1 + 1 + 1 - 1 = 3
```

dp[j] 表示：**填满j这么大容积的包，有dp[j]种方法**

递推公式：**dp[i]+=dp[i-nums[i]]**

10.一零和

dp数组的含义：**容量为j的背包所能装的最多物品数量**

递推公式：**dp[j] = max(dp[j], dp[j - values[i] + 1)**

11.完全背包

dp数组的含义：**容量为j的背包所能装的最大物品价值dp[j]（物品无限）**

递推公式：**dp[j]=max(dp[j],dp[j-weight[i]]+values[i])**

12.零钱兑换II

```
输入：amount = 5, coins = [1, 2, 5]
输出：4
解释：有四种方式可以凑成总金额：
5=5
5=2+2+1
5=2+1+1+1
5=1+1+1+1+1
```

dp[j]的含义是：**装满背包容量为j共有dp[j]中方法**

递推公式：**dp[j]+=dp[i-nums[i]]**

13.组合总和IV

```
输入：nums = [1,2,3], target = 4
输出：7
解释：
所有可能的组合为：
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)
请注意，顺序不同的序列被视作不同的组合。
```

dp数组的含义：**装满容量为j的背包的不同方法**

递推公式：**dp[j]+=dp[j-nums[i]]**

14.零钱兑换

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

dp数组的含义：**装满价值为j最少物品个数为dp[j]**

递推公式：**dp[j]=min(do[i-coins[i]]+1,dp[j]);**

15.完全平方数

```
输入：n = 12
输出：3
解释：12 = 4 + 4 + 4
```

dp[j]：**和为j的完全平方数的最少物品数量为dp[j]**

递推公式：**dp[j]=min(dp[j],dp[j-i*i]+1**

16.单词的拆分

```
输入: s = "leetcode", wordDict = ["leet", "code"]
输出: true
解释: 返回 true 因为 "leetcode" 可以由 "leet" 和 "code" 拼接成。
```

dp数组的含义：**字符串长度为i，如果dp[i]为true，则可以拆分为在字典中单词。**

递推公式：对于从**i到j的字串**，如果在**wordSet中查找**到这个字串并且**dp[i]为true**，dp[j]也为true（注意**初始化dp[0]=true**）

17.多重背包

将多重背包转换为01背包

```
for (int i = 0; i < nums.size(); i++) {
    while (nums[i] > 1) { // nums[i]保留到1，把其他物品都展开
        weight.push_back(weight[i]);
        value.push_back(value[i]);
        nums[i]--;
    }
}
```

18.打家劫舍I

```
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 。
```

dp数组的含义：**考虑下标i房间是否去偷的情况下，最多所能偷的钱为dp[j]**

递推公式：**dp[i]=max(dp[i-1],dp[i-2]+nums[i])**

19.打家劫舍II

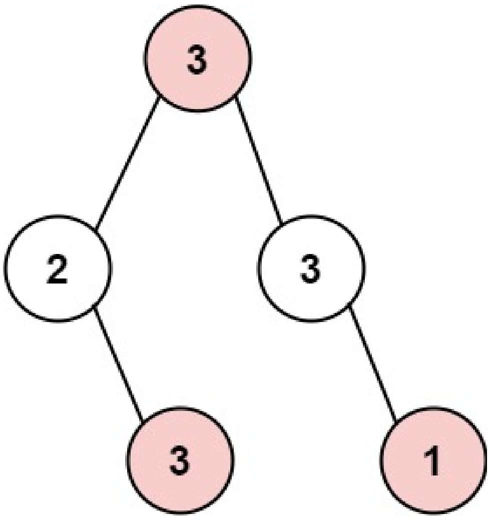
```
输入：nums = [2,3,2]
输出：3
解释：你不能先偷窃 1 号房屋（金额 = 2），然后偷窃 3 号房屋（金额 = 2）, 因为他们是相邻的。
```

注释：**将原数组拆分为去尾数组以及去头数组**，再调用I的函数

dp数组的含义：**考虑下标i房间是否去偷的情况下，最多所能偷的钱为dp[j]**

递推公式：**dp[i]=max(dp[i-1],dp[i-2]+nums[i])**

20.打家劫舍III




```
输入: root = [3,2,3,null,3,null,1]
输出: 7
解释: 小偷一晚能够盗取的最高金额 3 + 3 + 1 = 7
```

dp数组的含义：定义两个状态，由于递归的特性，**每一层都会存在这两个状态**

dp[0]表示不偷该节点的最大金钱

dp[1]表示偷该节点的最大金钱

```
class Solution {
private:
    vector<int> posttracking(TreeNode* cur) {
        if (cur == NULL) {//递归终止条件
            return vector<int>{0, 0};
        }

        vector<int> LeftDp = posttracking(cur->left);
        vector<int> RightDp = posttracking(cur->right);

        int Rob = cur->val + LeftDp[0] + RightDp[0];//偷当前的节点，其左右子节点不偷
        int NotRob = max(LeftDp[0], LeftDp[1]) + max(RightDp[0], RightDp[1]);//不偷
当前的节点，其左右子节点偷

        return vector<int>{NotRob, Rob};
    }
public:
    int rob(TreeNode* root) {
        vector<int> vec = posttracking(root);
        return max(vec[0], vec[1]);
```

```
    }
};
```

21.买卖股票的最佳时机I

```
输入：[7,1,5,3,6,4]
输出：5
解释：在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。
     注意利润不能是 7-1 = 6, 因为卖出价格需要大于买入价格；同时，你不能在买入前卖出股票。
```

dp数组的含义：

dp[i][0]第i天不持有股票的利润

dp[i][1]第i天持有股票的最大利润

递推公式：

dp[i][0] = max(dp[i - 1][0], dp[i - 1][1] + prices[i]);

dp[i][1] = max(dp[i - 1][1], -prices[i]);

22.买卖股票的最佳时机II

```
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 。
     总利润为 4 + 3 = 7 。
```

dp数组的含义：
dp[i][0]第i天不持有该股票的最大利润dp[i][1]第i天持有该股票的最大利润

递推公式：

dp[i][0] = max(dp[i - 1][0], dp[i - 1][1] + prices[i])
dp[i][1] = max(dp[i - 1][1], dp[i - 1][0] - prices[i])

23.买卖股票的最佳时机III

最多买入卖出两次

```
输入：prices = [3,3,5,0,0,3,1,4]
输出：6
解释：在第 4 天（股票价格 = 0）的时候买入，在第 6 天（股票价格 = 3）的时候卖出，这笔交易所能获得利润 = 3-0 = 3 。
     随后，在第 7 天（股票价格 = 1）的时候买入，在第 8 天 （股票价格 = 4）的时候卖出，这笔交易所能获得利润 = 4-1 = 3 。
```

dp[i][0]第i天没有进行任何操作的最大利润
dp[i][1]第i天买入了第1次的最大利润
dp[i][2]第i天卖出了第1次的最大利润
dp[i][3]第i天买入了第2次的最大利润
dp[i][4]第i天卖出了第2次的最大利润

递推公式：

dp[i][0] = dp[i-1][0]
dp[i][1] = max(dp[i - 1][1], dp[i - 1][0] - prices[i])
dp[i][2] = max(dp[i - 1][2], dp[i - 1][1] + prices[i])
dp[i][3] = max(dp[i - 1][3], dp[i - 1][2] - prices[i])

dp[i][4] = max(dp[i - 1][4], dp[i - 1][3] + prices[i])

24.买卖股票的最佳时机IV

```
输入: prices = [1,2,3,0,2]
输出: 3
解释: 对应的交易状态为: [买入, 卖出, 冷冻期, 买入, 卖出]
```

dp数组的含义：

dp[0][0]不操作

dp[i][1]第i天时，进行了第一次买入的最大利润

dp[i][2]第i天时，进行了第一次卖出的最大利润

dp[i][3]第i天时，进行了第二次买入的最大利润

dp[i][4]第i天时，进行了第二次卖出的最大利润

dp[i][5]第i天时，进行了第三次买入的最大利润

dp[i][6]第i天时，进行了第三次卖出的最大利润

....

递推公式：

```
for (int j = 0;j < 2 * k;j += 2) {//如果边界不好判断则带入k等于2进行判断
    dp[i][j + 1] = max(dp[i - 1][j + 1], dp[i - 1][j] - prices[i]);
    dp[i][j + 2] = max(dp[i - 1][j + 2], dp[i - 1][j + 1] + prices[i]);
}
```

25.有冷静期的买卖股票

```
输入: prices = [1,2,3,0,2]
输出: 3
解释: 对应的交易状态为: [买入, 卖出, 冷冻期, 买入, 卖出]
```

dp数组的含义：

dp[i][0] 第i天持有股票的最大利润

dp[i][1] 第i天保持股票卖出状态的最大利润

dp[i][2] 第i天卖出股票的最大利润

dp[i][3] 第i天为冷静期的最大利润

递推公式：

dp[i][0] = max(dp[i - 1][0], max(dp[i - 1][1], dp[i - 1][3]) - prices[i])
dp[i][1] = max(dp[i - 1][1], dp[i - 1][3])
dp[i][2] = dp[i - 1][0] + prices[i]
dp[i][3] = dp[i - 1][2]

26.有手续费的买卖股票

```
输入：prices = [1, 3, 2, 8, 4, 9], fee = 2
输出：8
解释：能够达到的最大利润:
在此处买入 prices[0] = 1
在此处卖出 prices[3] = 8
在此处买入 prices[4] = 4
在此处卖出 prices[5] = 9
总利润: ((8 - 1) - 2) + ((9 - 4) - 2) = 8
```

dp数组的含义：

dp[i][0]第i天不持有股票的最大利润

dp[i][1]第i天持有股票的最大利润

递推公式：

dp[i][0] = max(dp[i - 1][0], dp[i - 1][1] + prices[i] - fee)
dp[i][1] = max(dp[i-1][0]-prices[i],dp[i-1][1])

27.最长递增子序列

```
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 。
```

dp数组的含义为：**以数组中i为下标的最大递增子序列的长度**

递推公式：**dp[i] = max(dp[i], dp[j] + 1);**

28.最长连续递增序列

```
输入：nums = [1,3,5,4,7]
输出：3
解释：最长连续递增序列是 [1,3,5], 长度为3。
尽管 [1,3,5,7] 也是升序的子序列, 但它不是连续的，因为 5 和 7 在原数组里被 4 隔开。
```

dp数组的含义为：**以下标i为结尾的连续递增的子序列长度为dp[i]**

递推公式：**dp[i]=dp[i-1]+1**

29.最长连续重复子序列

```
输入：nums1 = [1,2,3,2,1], nums2 = [3,2,1,4,7]
输出：3
解释：长度最长的公共子数组是 [3,2,1] 。
```

dp数组的含义：**dp[i][j]是数组A以i-1为结尾，数组B以j-1为结尾的最大重复子数组的长度**

递推公式：

```
if (nums1[i-1] == nums2[j-1]) {
    dp[i][j] = dp[i - 1][j - 1] + 1;//i与j都往前倒退一位
}
```

30.最长公共子序列

```
输入：text1 = "abcde", text2 = "ace"
输出：3
解释：最长公共子序列是 "ace" ，它的长度为 3
```

dp数组的含义为：

dp[i][j]是数组A以i-1位置元素结尾，数组B以j-1位置元素结尾的最长公共子序列长度

递推公式：

```
if (text1[i - 1] == text2[j - 1]) {//如果成功匹配
    dp[i][j] = dp[i - 1][j - 1] + 1;//往前回退一位的情况+1
}
else {//如果没有成功匹配
    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);//之前就没有匹配(i没有匹配，j没有匹配)
}
```

31.最大连续子数组和

```
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 。
```

dp数组的含义：**当遍历到数组下标为i的时候，最大子数组和的值为dp[i]**

递推公式：**dp[i] = max(dp[i - 1] + nums[i], nums[i])**

32.判断s是否为t的子序列

```
输入：s = "abc", t = "ahbgdc"
输出：true
```

dp数组的含义为：

当遍历到i-1位置的s数组，j-1位置的t数组时，其最大公共子序列的长度。

递推公式：

```
if (s[i - 1] == t[j - 1]) {//如果匹配
    dp[i][j] = dp[i - 1][j - 1] + 1;
}
else {//如果不匹配：i不匹配、j不匹配
    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
}
```

33.不同的子序列

```
输入：s = "rabbbit", t = "rabbit"
输出：3
解释：
如下所示, 有 3 种可以从 s 中得到 "rabbit" 的方案。
rabbbit
```

```
rabbbit
rabbbit
```

dp数组的含义是：**当数组s遍历到i-1位置时，数组t遍历到j-1位置时的不同子序列的数量**

递推公式：

```
if (s[i - 1] == t[j - 1]) {//如果当前字母匹配
    dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];
    //dp[i-1][j-1]表示使用s[i-1]的情况
    //dp[i-1][j]表示使用s[i-2]的情况
}
else {//如果当前字母不匹配
    dp[i][j] = dp[i - 1][j];
}
```

34.两个字符串的删除操作

```
输入: word1 = "sea", word2 = "eat"
输出: 2
解释: 第一步将 "sea" 变为 "ea" , 第二步将 "eat "变为 "ea"
```

dp数组的含义：

当遍历到word1数组i-1位置，word2数组j-1位置时，使得word1==word2的最小删除步数

递推公式：

```
if (word1[i - 1] == word2[j - 1]) {
    dp[i][j] = dp[i - 1][j - 1];
}
else {
    dp[i][j] = min(dp[i - 1][j] + 1, min(dp[i][j - 1] + 1, dp[i - 1][j - 1] + 2));
}
```

35.编辑距离

```
输入：word1 = "horse", word2 = "ros"
输出：3
解释：
horse -> rorse (将 'h' 替换为 'r')
rorse -> rose (删除 'r')
rose -> ros (删除 'e')
```

dp数组的含义：**遍历数组word1到i-1位置，word2数组j-1的时候，让i-1及其以前的字符串与j-1及其以前的字符串相等的最小步骤**

推推公式：

```
if (word1[i - 1] == word2[j - 1]) {
    dp[i][j] = dp[i - 1][j - 1];//如果字母相等，则不需要进行操作
}
else {
    dp[i][j] = min(dp[i - 1][j] + 1, min(dp[i][j - 1] + 1, dp[i - 1][j - 1] + 1));
}
```

36.回文子串

```
输入：s = "aaa"
输出：6
解释：6个回文子串："a", "a", "a", "aa", "aa", "aaa"
```

dp数组的含义为：**从下标i到下标j的子串是否为回文串**

递推公式：

```
if (s[i] == s[j]) {
    if (j - i <= 1) {//a或者aa的情况
            dp[i][j] = true;
            result++;
    }
    else if (dp[i + 1][j - 1] == true) {//aaa的情况
        dp[i][j] = true;
        result++;
    }
]
```

37.最长回文子序列

```
输入：s = "bbbab"
输出：4
解释：一个可能的最长回文子序列为 "bbbb" 。
```

dp数组的含义：**dp[i][j]从下标i到下标j的最长回文子序列**

递推公式：

```
if (s[i] == s[j]) {//如果两端字母相同
    dp[i][j] = dp[i + 1][j - 1] + 2;//前一种情况加上左右两端的长度
}
else {//如果两端字母不同,则让i进一位或者j退一位，取最大值即可
    dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
}
```