动态规划-打家劫舍 专题

打家劫舍

力扣题目链接

房间排列是线性的。

`dp[i]` 数组含义：在第 i 间屋子所偷的最大金钱为 `dp[i]`

递推公式：

只需要如果获取 dp[i]。

- 如果**考虑**偷 第 i 间屋子的金钱，那么`dp[i] = dp[i-2] + nums[i]` （即 到了第 i-2 间房子的时候偷的最大金钱 `dp[i-2]` + 当下第 i 间房子的金钱 `nums[i]` ）
- 前一间房子被偷了金钱，那么 第 i 间就不能偷了。`dp[i] = dp[i-1]`

`dp[i]` = Math.max(`dp[i-2] + nums[i]` , `dp[i-1]`);

初始化数值：

`dp[0]` 即 为 `nums[0]` ;

`dp[1]` 即第一间房子和第二间房子看看谁家的现金最多就考虑偷谁的，即 `dp[1] = Math.max(nums[0],nums[1])`

```
public int rob(int[] nums) {
        if (nums.length == 1) return nums[0];
        // dp数组：偷到第j个房间时偷到的最大现金总和是dp[j]
        int[] dp = new int[nums.length];
        dp[0] = nums[0];
        dp[1] = Math.max(dp[0], nums[1]);
        for (int i = 2; i < nums.length; i++) {
            dp[i] = Math.max(dp[i - 2] + nums[i], dp[i - 1]);
        }
        return dp[dp.length - 1];
    }
/*
时间复杂度：O(n)
空间复杂度：O(n)
*/
```

打家劫舍II

力扣题目链接

房屋排列是环状的了。

分三种情况：

1. 不偷最后一间房子的；那么数组的长度就是 **[0,nums.length - 1];**
2. 不偷第一间房子的，那么数组的长度就是 **[1,nums.length];**
3. 不偷前后的房子，那么数组的长度就是 **[1,nums.length-1];**

因为第1和第2种情况包含了 第3种情况，那么就可以直接算出第一和第二种情况即可。

```
// 代码的顺序不一样，但是思路一样
public int rob(int[] nums) {
        if (nums.length == 0) return 0;
        if (nums.length == 1) return nums[0];
        int result1 = robRange(nums, 0, nums.length - 2); // 情况二
        int result2 = robRange(nums, 1, nums.length - 1); // 情况三
        return Math.max(result1, result2);
    }

    int robRange(int[] nums, int start, int end) {
        if (start == end) return nums[start];

        int[] dp = new int[nums.length];
        dp[start] = nums[start];
        dp[start + 1] = Math.max(nums[start + 1], nums[start]);
        for (int i = start + 2; i < end; i++) {
            dp[i] = Math.max(dp[i - 2] + nums[i], dp[i - 1]);
        }
        return dp[dp.length - 1];
    }
/*
时间复杂度：O(n)
空间复杂度：O(n)
*/
```

打家劫舍III

力扣题目链接

这道题目瞬间变难了，树形的房屋排列。

结合递归三部曲和动归五部曲来完成

1. 确定递归函数的返回值和参数 `public int[] rotTree(TreeNode root)`

   这里我们要求的是 **一个结点偷与不偷两个状态下 所得的最大金钱**，那么就需要长度为2的dp数组即可。所以 `dp[0]` ：下标为0记录不偷该结点所得到的最大金钱；`dp[1]` 下标为1为偷该结点所得的最大金钱；

2. 确定终止条件

   如果遇到空结点，无论偷还是不偷，金钱所得都为0；

   ```
   if(root == null) return res;
   ```

   这也相当于数组的初始化了。

3. 确定遍历顺序

   后序遍历，因为需要递归函数的返回值来做下一步的计算。

   递归左结点，得到左结点偷与不偷的金钱；

   递归右结点，得到右结点偷与不偷的金钱；

```
// 0：不偷  1：偷
int[] left = rotTree(root.left);
int[] right = rotTree(root.right);
```

4. 确定单层的递归逻辑

   如果是**偷当前结点**：那么左右孩子不能偷，`int val1 = root.left.val + left[0] + right[0];`

   如果是**不偷当前节点**：那么左右孩子考虑偷，且需要找到最大值，`int val2 = Math.max(left[0],left[1]) + Math.max(right[1],right[0]);`

```
public int rob3(TreeNode root) {
        int[] res = robAction1(root);
        return Math.max(res[0], res[1]);
    }

    int[] robAction1(TreeNode root) {
        int res[] = new int[2];
        if (root == null)
            return res;

        int[] left = robAction1(root.left);
        int[] right = robAction1(root.right);

        res[0] = Math.max(left[0], left[1]) + Math.max(right[0], right[1]);
        res[1] = root.val + left[0] + right[0];
        return res;
    }
/*
时间复杂度：O(n)，每个节点只遍历了一次
空间复杂度：O(log n)，算上递推系统栈的空间
*/
```

空间复杂度的计算方法：每次递归的空间复杂度 * 递归深度