# 动态规划

# 写在前面：

递归是一种变成方法或者技巧。

动态规划，深搜，回溯都可以用递归，其实本质上也都是属于穷举所有可能，特别是深搜和回溯，穷尽各种可能的意图特别明显。

这三个涉及到递归，那么实现起来都可以用如下的方式去想，特别是回溯和深搜。

Step是什么（这一步是干什么），有什么选择（可能是左右子树，可能是for循环8个方向，或者N叉树的多节点）。

（·）回溯：【可看对应文档】

一般有vis数组进行标记，标记抹除，以及通过函数参数进行加减运行，录入curSum + nums[index].

（2）深搜【可看对应文档】：

基本和回溯类似。

但是深搜一般分为自上到下，以及自底向上。

例如求树的最大深度，可以利用自顶向下的方法（需要父节点传参数下来），也可以通过自底向上，一般每个选择都有临时变量进行保存，然后进行运算或逻辑判断，从而得到当前状态的值或者最优结果。自底向上的深搜基本可以认为是动态归化的初版。

深搜有时是非常简单的利用例如染色。

（3）动态规划【此文档】

其实是由这么一个过程得到的。

递归的暴力解法（深搜，但是是自底向上的形式）🡪带备忘录cache的递归解法（去除重复计算的子问题）->非递归的动态规划解法。带备忘录的自底向上的递归几乎和迭代的动态规划复杂度是一样的。

一旦练得多连得多是可以直接写出非递归的动态规划的。

一般形式是求最值。

动态规划的问题一定具备最优子结构。即最优的结果可以通过子问题的最优解得到。

一般先利用穷举，找出状态方程，这一个是最难的，但有一个思维框架。

明确状态—定义dp数组/函数的定义->明确选择->明确初始值即base case.

## 1.[213. 打家劫舍 II](https://leetcode-cn.com/problems/house-robber-ii/)【深搜超时，自底向上的递归即动态规划可行】



思路1：深搜 超时，并且自上到下的如果到树那里，这个题就不好写了。

Step是选择一个元素，选择是这个这个元素是否要叠加到当前的和上，递归结束时，判断当前结果curSum即g\_max比较大小。另外使用了vis数组，仅仅用于判断在nums[0]选择时，不选最后一个。

int g\_max = 0;

int\* vis = NULL;

void PickOneRommDfs(int\* nums, int num, int curStartIndex, int curSum)

{

    int i;

    if (g\_max < curSum) {

        g\_max = curSum;

    }

    if (curStartIndex >= num) {

        return;

    }

    /\*\*不用for循环了因为只能从curStartIndex中选择，然后就是选或不选的问题\*/

        /\*\*选\*/

    vis[curStartIndex] = 1;

    if (!(curStartIndex == num - 1 && vis[0] == 1)) {

        PickOneRommDfs(nums, num, curStartIndex + 2, curSum + nums[curStartIndex]);

    }

    vis[curStartIndex] = 0;

        /\*\*不选\*/

    PickOneRommDfs(nums, num, curStartIndex + 1, curSum);

}

int rob(int\* nums, int numsSize)

{

    int i;

    g\_max = 0;

    vis = (int\*)calloc(numsSize, sizeof(int));

    if (numsSize == 1) {

        return nums[0];

    }

    PickOneRommDfs(nums, numsSize, 0, 0);

    free(vis);

    return g\_max;

}

思路二：

**解决动态规划问题就是找「状态」和「选择」，仅此而已**。

**你面前房子的索引就是状态，抢和不抢就是选择**。在两个选择中，每次都选更大的结果，最后得到的就是最多能抢到的 money。是后序遍历的典型。

// 计算当前的状态的startIndex时的最右选择

int PickOneRoomDfs(int\* nums, int startIndex, int endSize)

{

    int doIt, notDoIt;

    int res;

    if (startIndex >= endSize) {

        return 0;

}

// 去重复计算

    if (memMax[startIndex] != -1) {

        return memMax[startIndex];

    }

    notDoIt = PickOneRoomDfs(nums, startIndex + 1, endSize);

    doIt = PickOneRoomDfs(nums, startIndex + 2, endSize) + nums[startIndex];

res = notDoIt > doIt ? notDoIt :  doIt;

// 进行cache缓存，备忘录

    memMax[startIndex] = res;

    return res;

}

int rob(int\* nums, int numsSize)

{

    int i;

memMax = (int\*)calloc(numsSize, sizeof(int));

// 分2次并取最大值

    for (i = 0; i < numsSize; i++) {

        memMax[i] = -1;

    }

    int doFirst = PickOneRoomDfs(nums, 0, numsSize - 1);

    for (i = 0; i < numsSize; i++) {

        memMax[i] = -1;

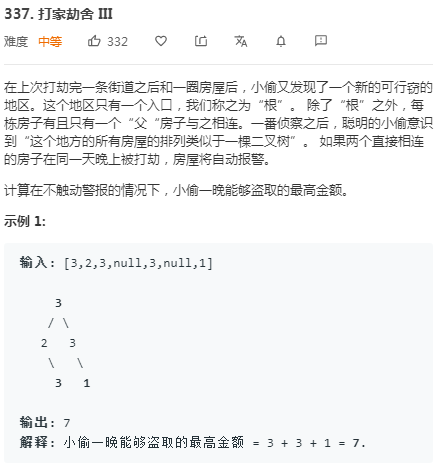
    }

    int notDoFirst = PickOneRoomDfs(nums, 1, numsSize);

    return doFirst > notDoFirst ? doFirst : notDoFirst;

}

## 2. [337. 打家劫舍 III](https://leetcode-cn.com/problems/house-robber-iii/)【自底向上，没有加备忘录其实只属于深搜】



 int Dfs(struct TreeNode\* root)

 {

    int doIt, notDoIt;

    if (root == NULL) {

        return 0;

    }

    // 抢

    doIt = (root->left == NULL ? 0 : (Dfs(root->left->left) + Dfs(root->left->right))) +

        (root->right == NULL ? 0 : (Dfs(root->right->left) + Dfs(root->right->right)));

    // 不抢

    notDoIt = Dfs(root->left) + Dfs(root->right);

    doIt += root->val;

    return (doIt > notDoIt ? doIt : notDoIt);

 }

int rob(struct TreeNode\* root)

{

    if (root == NULL) {

        return 0;

    }

    return Dfs(root);

}