【第二十七周】动态规划算法优化

1、494. 目标和

0/1背包

f[i][j] = f[i-1][j] + f[i][j-x] 忽略了数字出现的先后顺序

f[j] 使用所有硬币拼凑出来金额为j的方法总数

dp[j] 使用数组nums里面的所有数字的或者加/减，得出的运算结果满足目标值target的排列组合/方法总数

dp[j] = n/2(j-x),x每一种硬币的面值，j = 1

```
/**
 * @param {number[]} nums
 * @param {number} target
 * @return {number}
 */
var findTargetSumWays = function(nums, target) {
    let sum = nums.reduce((t,v) => t + v);
    if((sum - target) % 2 !== 0 || sum - target < 0) return 0;
    let result = parseInt((sum - target) / 2);
    let dp = new Array(result + 1).fill(0);
    dp[0] = 1;
    let n = nums.length;
    for(let i = 0; i < n;i++){
        for(let j = result;j >= nums[i];j--){
            dp[j] = dp[j] + dp[j-nums[i]];
        }
    }
    return dp[result];
};
```

2、518. 零钱兑换 II

1、凑成总金额所需的 方法总数

2、dp[i][j]等于什么?

3、dp[i][j]：使用了前i个硬币，拼凑j元钱数量

4、dp[i][j] = dp[i-1][j] + dp[i][j-x];x:代表第i硬币的金额

5、dp[i-1][j]：使用了第i种硬币凑出来的方法

6、dp[i][j-x]：没有用了第i种硬币凑出来的方法

```
/**
 * @param {number} amount
 * @param {number[]} coins
 * @return {number}
 */
var change = function(amount, coins) {
    const dp = new Array(amount + 1).fill(0);
    dp[0] = 1;
    // 正向刷表
    for(const x of coins){
        for(let j = x;j <= amount;j++){
            dp[j] += dp[j - x]
        }
    }
    return dp[amount];
};
```

3、377. 组合总和 IV

题意要求计算并返回，部分数字元素和值为目标值的所有可能情况

1、初始化 dp[0]=1；

2、遍历 i 从 1 到 target，对于每个 i，进行如下操作：

遍历数组 nums 中的每个元素 num，当 num≤i 时，将 dp[i−num] 的值加到 dp[i]。

3、最终得到 dp[target] 的值即为答案。

```
/**
 * @param {number[]} nums
 * @param {number} target
 * @return {number}
 */
var combinationSum4 = function(nums, target) {
    const dp = new Array(target + 1).fill(0);
    dp[0] = 1;
    for(let i =1;i <= target;i++){
        for(const num of nums){
            if(num <= i){
                dp[i] += dp[i - num];
            }
        }
    }
```

```
    return dp[target];
};
```

4、382. 链表随机节点

蓄水池做法：一次性循环，随即取到的数据概率是一样的

定链表的长度n，随机抽取目标节点target，开始继续循环遍历

1、访问到第1个节点的时候，我们在[1,1]选取节点，第一个节点就是目标节点target： 1

2、访问到第2个节点的时候，我们在[1,2]选取节点，第2个节点就是目标节点target： 2

3、访问到第3个节点的时候，我们在[1,3]选取节点，第3个节点就是目标节点target： 3

访问.....

最后访问到第n个节点的时候，我们在[1,n]选取节点，第一个节点就是目标节点target： n

也就是说最后的我目标节点应该是索引为n的节点

```
/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param head The linked list's head.
        Note that the head is guaranteed to be not null, so it contains at
least one node.
 * @param {ListNode} head
 */
var Solution = function(head) {
    this.head = head;
};

/**
 * Returns a random node's value.
 * @return {number}
 */
Solution.prototype.getRandom = function() {
    let res = null,num = 0;
    let head = this.head;
    while(head != null){
        num++;
        if(!Math.floor(Math.random() * num)) res = head.val;
        head = head.next;
```

```
    }
    return res;
};

/**
 * Your Solution object will be instantiated and called as such:
 * var obj = new Solution(head)
 * var param_1 = obj.getRandom()
 */
```

5、462. 最少移动次数使数组元素相等 II

因为我们求的是最少的移动次数，所以说我们的值应该是往中间靠拢，所以就是求的这个数组里面的中位数

```
/**
 * @param {number[]} nums
 * @return {number}
 */
//  找到一个中位数
var minMoves2 = function(nums) {
    nums.sort((a,b) => a-b);
    let avg = nums[Math.ceil(nums.length / 2) - 1];
    return nums.reduce((total,nums) =>{
        return total + Math.abs(nums - avg);
    },0);
};
```

6、77. 组合

1、题意要求：返回范围 [1, n] 所有长度为k的子组合

2、查找的过程看成一棵树，去找每一条路径，路径长度是k

3、用到的算法：【dfs】

```
/**
 * @param {number} n
 * @param {number} k
 * @return {number[][]}
 */
var combine = function(n, k) {
    const ans = [];
    const dfs = (cur,n,k,temp) =>{
```

```
        // 减枝 ：  temp的长度加上区间[cur,n]的长度小于k,不可能构造出来长度为k的排列组合
temp
        if(temp.length + (n - cur + 1) < k){
            return;
        }
        // 记录合法的答案
        if(temp.length == k){
            ans.push(temp);
            return;
        }
        // 考虑选择当前的位置
        dfs(cur + 1,n,k,[...temp,cur]);
        // 不考虑当前的位置
        dfs(cur + 1,n , k,temp);
    }
    dfs(1,n,k,[]);
    return ans;

};
```

7、234. 回文链表

1、回文链表就是，从头往后扫描到的结果和从后往前扫描的结果是一样的。

2、使用快慢指针寻找中位数。快慢指针同时在链表头部出发，快指针每次走两步，慢指针每次走一步，当快指针走到链表的尾部的时候，慢指针正好落在链表的最中间。

3、反转前半部分的链表看看是否和后半部分一致，一致就是回文链表。

```
 //  搞快慢指针 快指针到尾部 慢指针刚好遍历完前半部分
var isPalindrome = function(head) {
    let fast = head;
    let slow = head;//快慢指针用于让slow指向链表后半部分的开头
    let pre = head, prepre = null;//反转指针的原理 详见leetcode206
    while(fast != null && fast.next != null){
        //01 快慢指针 获得链表
        fast = fast.next.next;
        slow = slow.next;
        //02 反转前半部分链表
        pre.next = prepre;
        prepre = pre;
        pre = slow;//slow指向的正好是pre下一步要指向的位置
    }
    //如果结点数为奇数个 slow要再往下指一位（具体看图）
    if(fast != null){
        slow = slow.next;
    }
```

```
    //接下来比较前半部分的反转和后半部分即可
    while(slow != null){
        if(slow.val != prepre.val){
            return false;
        }
        slow = slow.next;
        prepre = prepre.next;
    }
    return true;
};
```