【第二十三周】手撕红黑树(下)-删除调整

1、528. 按权重随机选择

方法：前缀和 + 二分查找

1、设数组 w 的权重之和为 total。根据题目的要求，我们可以看成将 [1,total] 范围内的所有整数分成 n 个部分（其中 n 是数组 w 的长度），第 i 个部分恰好w[i] 个整数，并且这n个部分两两的交集为空。随后我们[1,total] 范围内随机选择一个整数x，如果整数x被包含在第 i 个部分内，我们就返回 i。

2、一种较为简单的划分方法是按照从小到大的顺序依次划分每个部分。如果我们用pre[i] 表示数组 w 的前缀和：第 i 个区间的左边界就是 pre[i]−w[i]+1，右边界就是pre[i]。由于pre[i] 是单调递增的，因此我们可以使用二分查找在 O(logn) 的时间内快速找到 i，即找出最小的满足x≤pre[i] 的下标 i。

```
var Solution = function(w) {
    pre = new Array(w.length).fill(0);
    pre[0] = w[0];
    for (let i = 1; i < w.length; ++i) {
        pre[i] = pre[i - 1] + w[i];
    }
    this.total = _.sum(w);
};

Solution.prototype.pickIndex = function() {
    const x = Math.floor((Math.random() * this.total)) + 1;
    const binarySearch = (x) => {
        let low = 0, high = pre.length - 1;
        while (low < high) {
            const mid = Math.floor((high - low) / 2) + low;
            if (pre[mid] < x) {
                low = mid + 1;
            } else {
                high = mid;
            }
        }
        return low;
    }
    return binarySearch(x);
};
```

2、382. 链表随机节点

蓄水池考虑的是如何循环一次随机取出对应的值

1、链表长度为n，随机抽取的目标节点为target，我们循环访问这个链表。

2、访问到第1个节点，我们则在[1,1]选取节点，第1个节点则为目标节点target。

3、访问到第2个节点，我们则在[1,2]选取节点，如果随机选择的节点是2，则目标节点为索引2的节点，否则节点不变。

4、访问到第3个节点，我们则在[1,3]选取节点，如果随机选择的节点是3，则目标节点为索引3的节点，否则节点不变。

5、访问到.....

6、访问到第n个节点，我们则在[1,n]随机选取节点，如果随机选择的节点是n,则目标节点为索引n的节点，否则节点不变 。

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @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 num = 0
    let res = null
    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()
 */
```

3、462. 最少移动次数使数组元素相等 II

总共的步数最小，所以数往中间走，所以本题的核心在于找到中位数

```
var minMoves2 = function(nums) {
    nums.sort((a, b) => a - b);
    let avg = nums[Math.ceil(nums.length/2) - 1];
    return nums.reduce((total, num) => {
        return total+Math.abs(num - avg)
    },0)
};
```

4、77. 组合

我们把这个过程当成一个树，这样可以考虑用DFS来解，就是求每条路径，

```
var combine = function(n, k) {
    const ans = [];
    const dfs = (cur, n, k, temp) => {
        // 剪枝：temp 长度加上区间 [cur, n] 的长度小于 k，不可能构造出长度为 k 的 temp
        if (temp.length + (n - cur + 1) < k) {
            return;
        }
        // 记录合法的答案
        if (temp.length == k) {
            ans.push(temp);
            return;
        }
        // 考虑选择当前位置
        dfs(cur + 1, n, k, [...temp, cur]);
        // 考虑不选择当前位置
        dfs(cur + 1, n, k, temp);
    }
    dfs(1, n, k, []);
    return ans;
};
```

5、234. 回文链表

方法：递归

1、currentNode 指针是先到尾节点，由于递归的特性再从后往前进行比较。frontPointer 是递归函数外的指针。若 currentNode.val != frontPointer.val 则返回 false。反之，frontPointer 向前移动并返回 true。

2、算法的正确性在于递归处理节点的顺序是相反的（回顾上面打印的算法），而我们在函数外又记录了一个变量，因此从本质上，我们同时在正向和逆向迭代匹配。

```
 let frontPointer;

const recursivelyCheck = (currentNode) => {
    if (currentNode !== null) {
        if (!recursivelyCheck(currentNode.next)) {
            return false;
        }
        if (currentNode.val !== frontPointer.val) {
            return false;
        }
        frontPointer = frontPointer.next;
    }
    return true;
}

var isPalindrome = function(head) {
    frontPointer = head;
    return recursivelyCheck(head);
};
```