递归和快速排序

目标：对采用递归思想解决问题和自己分析算法的复杂度更有信心。

目录

- 递归的基本思想
- 递归的解题思路及其实例
- 递归的复杂度分析
- 快速排序
- 聊聊刷Leetcode题

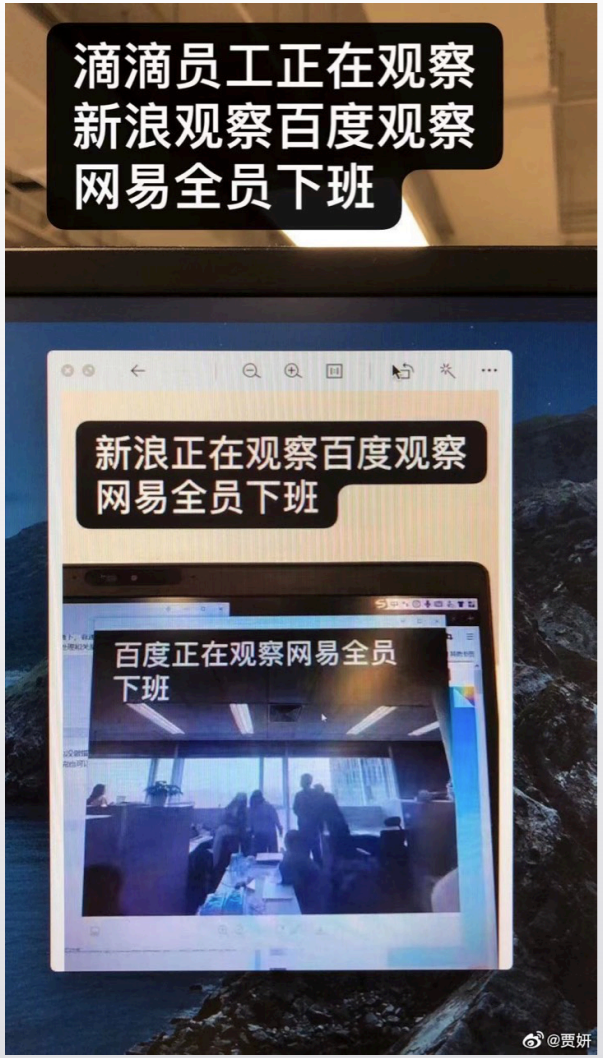
递归的基本思想

- 什么是递归？

从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？

“从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？

‘从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？……’”



递归算法的定义

- **方法自己调用自己**

例如：f(x)=f(a)+f(b) f 代表函数

看看真实的递归

例子：1+2+3+4……+100；

两种实现：

1. For 循环
2. 递归实现

递归的解题思路

字符串反转（来源leetcode: 344. *https://leetcode-cn.com/problems/reverse-string/* ）

编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 char[] 的形式给出。

不要给另外的数组分配额外的空间，你必须原地修改输入数组、使用 O(1) 的额外空间解决这一问题。

你可以假设数组中的所有字符都是 ASCII 码表中的可打印字符。

示例 1：

输入：["h","e","l","l","o"]

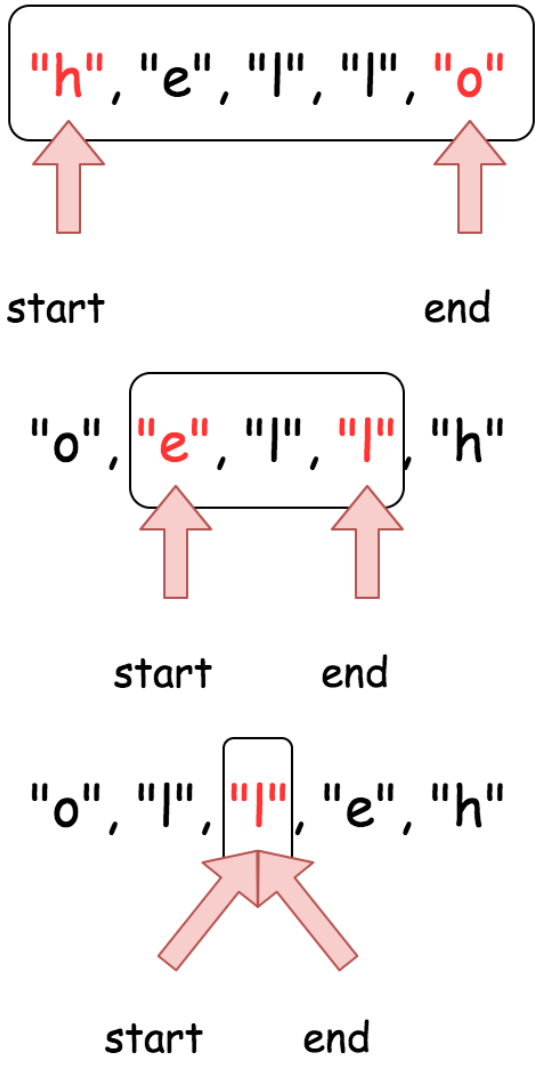
输出：["o","l","l","e","h"]

示例 2：

输入：["H","a","n","n","a","h"]

输出：["h","a","n","n","a","H"]

字符串反转



1. Problem: deal with a string "hello".
Swap and move pointers.

2. Subproblem : deal with a string "ell".
Swap and move pointers.

3. Subproblem: deal with a string "l".
start = end --> base case.

递归的解题思路

递归程序的基本步骤：

(1) 初始化算法。递归程序通常需要一个开始时使用的种子值（seed value）。要完成此任务，可以向函数传递参数，或者提供一个入口函数，这个函数是非递归的，但可以为递归计算设置种子值。

(2) 检查要处理的当前值是否已经与基线条件相匹配。如果匹配，则进行处理并返回值。

(3) 使用更小的或更简单的子问题（或多个子问题）来重新定义答案。

(4) 对子问题运行算法。

(5) 将结果合并入答案的表达式。

(6) 返回结果。

递归的解题思路

- **递归出口(终止递归的条件)**
- **递归表达式(规律)**

递归中的重复计算问题

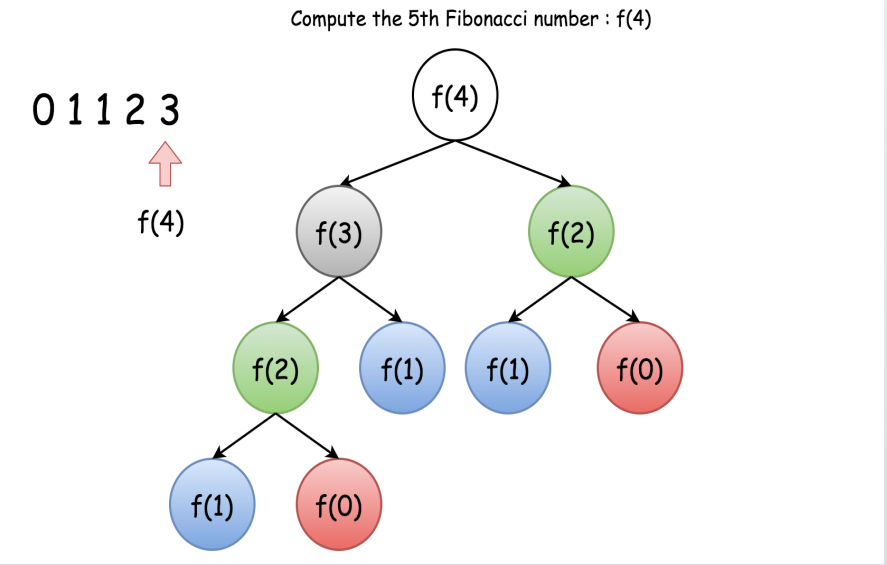
- 经典递归

斐波那契数，通常用F(n)表示，形成的序列称为斐波那契数列。该数列由0和1开始，后面的每一项数字都是前面两项数字的和。也就是：

*

* *F(0) = 0，F(1) = 1*
* *F(n) = F(n - 1) + F(n - 2)，其中n > 1*
* *给你n，请计算F(n)。*

- F(4)是多少，你可以应用上面的公式并进行展开：
- F(4) = F(3) + F(2) = (F(2) + F(1)) + F(2)



记忆化

我们将进一步研究递归可能出现的重复计算问题。 然后我们将提出一种常用的技术，称为记忆化（memoization），可以用来避免重复计算问题

记忆化是一种优化技术，主要用于**加快**计算机程序的速度，方法是**存储**昂贵的函数调用的结果，并在相同的输入再次出现时返回缓存的结果

简单点：就是用存储来存中间值，避免重复计算；

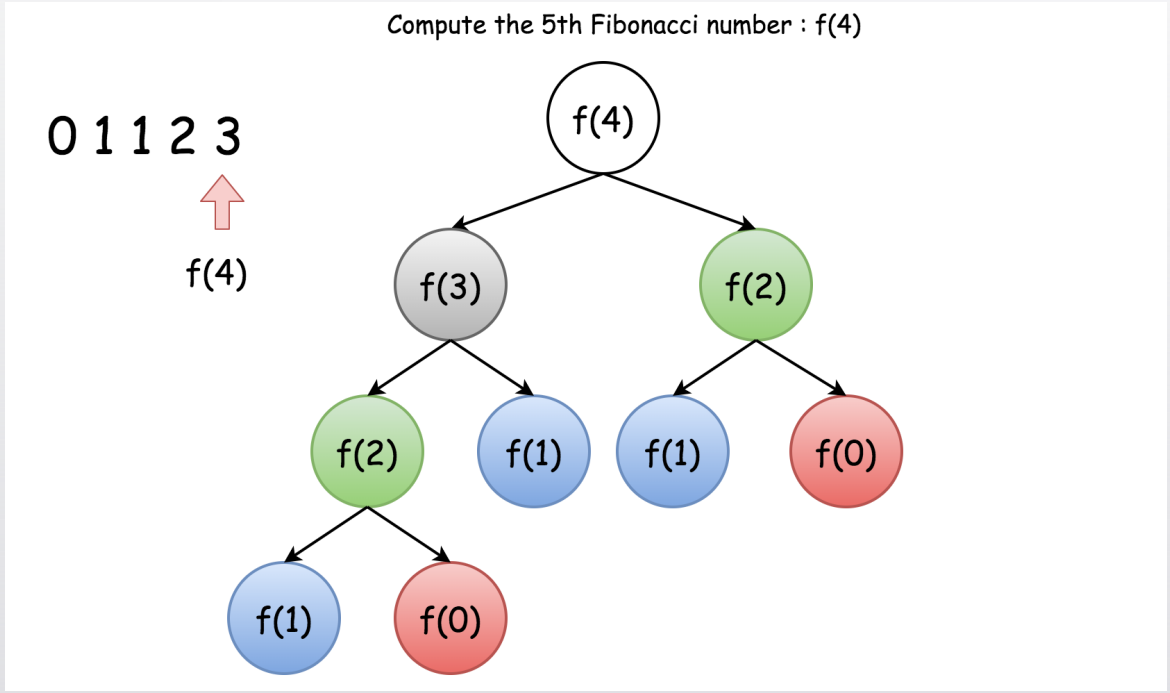
递归的复杂度分析

1.时间复杂度分析

2.空间复杂度分析

时间复杂度分析

执行树



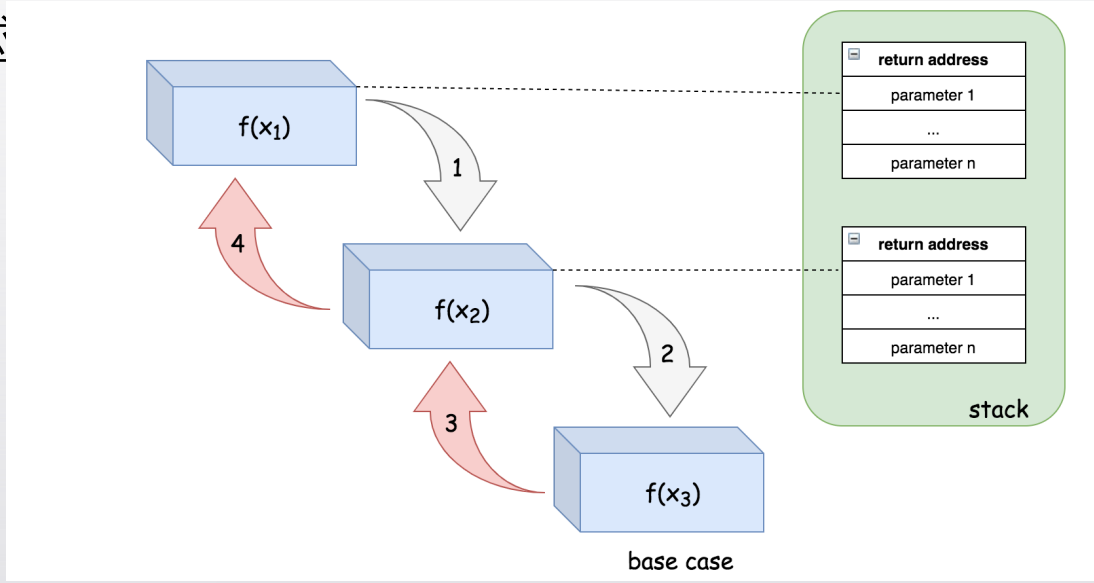
斐波那契数列: $O(2^n)$.
用记忆化方法：O(*n*)
字符串反转：O(*n*)
从1到n的求和：o(n)

时间复杂度分析：主要递归调用中每次调用几次自身。如果是一次，那就是o(n),如果是两次那就是 $O(2^n)$.（最坏）

空间复杂度分析

递归相关空间（recursion related space）

- 函数调用的返回地址。一旦函数调用完成，程序应之前的点；
- 传递给函数调用的参数；
- 函数调用中的局部变量。.



非递归相关空间（non-recursion related space）

指的是与递归过程没有直接关系的内存空间，通常包括为全局变量分配的空间（通常在堆中）。

快速排序

快速排序的核心思想

找到基准值的位置

快速排序的步骤

- 第一步：选择一个值作为基准值
- 第二步：分割基准值两边的元素，小于基准值的放在左边，大于基准值的放在右边
- 第三步：对基准值左右两侧进行递归操作第二步。

快速排序的步骤

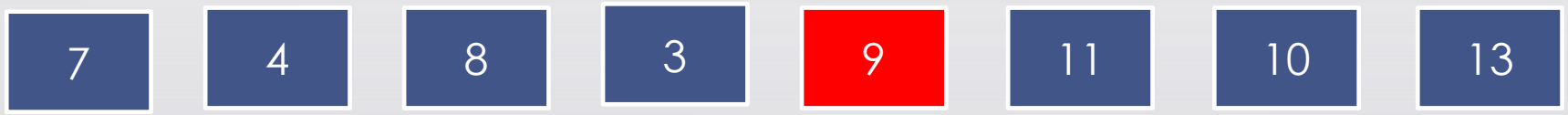
- 第一步：选择一个基准值

基准值

快速排序的步骤

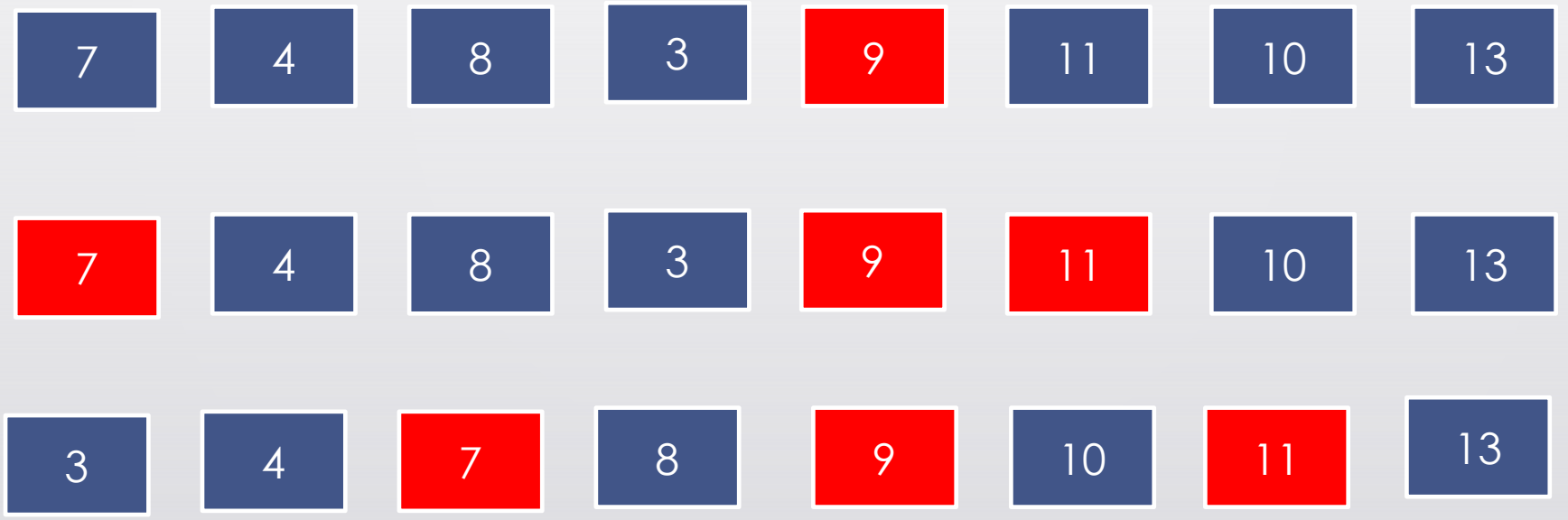
- 第二步：分割基准值两边的元素，小于基准值的放在左边，大于基准值的放在右边

基准值9



快速排序的步骤

- 第三步：分割基准值两边的元素，小于基准值的放在左边，大于基准值的放在右边



代码实现

```
/**
 * 快速排序的实现
 * @param nums  需要排序的数组
 * @param left  开始下标
 * @param right   结束下标
 * @return
 */
public static void quickSort(int[] nums, int left, int right){

    //判断是不是结束
    if(left >=right){
        return;
    }

    //1.选取基准值
    int key = nums[left];
    int i = left;
    int j = right;

    int emptyIndex = i;

    //2.左右分割
    while (i<j){

        //从右往左找，小于基准的的元素
        while (i<j && nums[j]>=key){
            j--;
        }

        if(i<j){
            nums[emptyIndex]  = nums[j];
            emptyIndex = j;
        }

        //从左往右找，大于基准的元素
        while(i<j && nums[i] <=key){
            i++;
        }

        if(i<j){
            nums[emptyIndex] = nums[i];
            emptyIndex =i;
        }

    } //左右分割完成

    nums[emptyIndex] = key;

    //第三步
    //左侧排序
    quickSort(nums,left,i-1);
    //右侧排序
    quickSort(nums,i+1,right);


}
```

复杂度分析

- 空间：最优：O(logn)，最差O(n)
- 时间：最优：O(nlogn)。，最差O(n^2)

聊聊刷Leetcode 题

1. 面试算法题的要求

   15分钟内完成一条算法题的编写（考察思维，基础，动手能力，解决问题的能力）

   NO IDEA ,没有提示，class, main函数都要手写。

2. 如何刷算法题。

   三遍法

   类型法

   错题法

   用白纸写算法

Leetcode ：70 爬楼梯

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

注意：给定 n 是一个正整数。

示例 1：

输入： 2

输出： 2

解释： 有两种方法可以爬到楼顶。

1. 1 阶 + 1 阶

2. 2 阶

示例 2：

输入： 3

输出： 3

解释： 有三种方法可以爬到楼顶。

1. 1 阶 + 1 阶 + 1 阶

2. 1 阶 + 2 阶

3. 2 阶 + 1 阶

总结

递归是一门伟大的艺术，使得程序的正确性更容易确认，而不需要牺牲性能，但这需要程序员以一种新的眼光来研究程序设计。对新程序员 来说，命令式程序设计通常是一个更为自然和直观的起点，这就是为什么大部分程序设计说明都集中关注命令式语言和方法的原因。 不过，随着程序越来越复杂，递归程序设计能够让程序员以可维护且逻辑一致的方式更好地组织代码。

作业

按照我的方法刷leetcode;

1. 二叉树的最大深度 （104） ：https://leetcode-cn.com/problems/maximum-depth-of-binary-tree/