王争的算法训练营

习题课：递归和分治

配套习题（8）：

剑指 Offer 10- I. 斐波那契数列 （简单）

剑指 Offer 10- II. 青蛙跳台阶问题（简单）

面试题 08.01. 三步问题 （简单）

剑指 Offer 06. 从尾到头打印链表 （简单）

剑指 Offer 24. 反转链表 （简单）

剑指 Offer 25. 合并两个排序的链表 （中等）

剑指 Offer 16. 数值的整数次方 （中等）

面试题 08.05. 递归乘法 （中等）

习题课：递归和分治

王争的算法训练营

剑指 Offer 10- I. 斐波那契数列 （简单）

写一个函数，输入 `n` ，求斐波那契（Fibonacci）数列的第 `n` 项（即 `F(N)`）。斐波那契数列的定义如下：

```
F(0) = 0,   F(1) = 1
F(N) = F(N - 1) + F(N - 2), 其中 N > 1.
```

斐波那契数列由 0 和 1 开始，之后的斐波那契数就是由之前的两数相加而得出。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

```
输入：n = 2
输出：1
```

示例 2：

```
输入：n = 5
输出：5
```

提示：

- `0 <= n <= 100`

```java
class Solution {
    private int mod = 1000000007;
    public int fib(int n) {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return (fib(n-1)+fib(n-2))%mod;
    }
}
```

执行结果： 超出时间限制 显示详情 >

最后执行的输入：

43

时间复杂度：？

空间复杂度：？

```java
class Solution {
    private int mod = 1000000007;
    private int[] memo;

    public int fib(int n) {
        memo = new int[n+1];
        return fib_r(n);
    }

    private int fib_r(int n) {
        if (n == 0) return 0;
        if (n == 1) return 1;
        if (memo[n] != 0) return memo[n];
        memo[n] = (fib_r(n-1)+fib_r(n-2))%mod;
        return memo[n];
    }
}
```

时间复杂度：？

空间复杂度：？

剑指 Offer 10- II. 青蛙跳台阶问题（简单）

一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个 `n` 级的台阶总共有多少种跳法。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

```
输入：n = 2
输出：2
```

示例 2：

```
输入：n = 7
输出：21
```

示例 3：

```
输入：n = 0
输出：1
```

提示：

- `0 <= n <= 100`

```java
class Solution {
    private int mod = 1000000007;
    private Map<Integer, Integer> memo = new HashMap<>();
    public int numWays(int n) {
        if (n == 0) return 1;
        if (n == 1) return 1;
        if (memo.containsKey(n)) {
            return memo.get(n);
        }
        int ret = (numWays(n-1)+numWays(n-2))%mod;
        memo.put(n, ret);
        return ret;
    }
}
```

面试题 08.01. 三步问题 （简单）

三步问题。有个小孩正在上楼梯，楼梯有n阶台阶，小孩一次可以上1阶、2阶或3阶。实现一种方法，计算小孩有多少种上楼梯的方式。结果可能很大，你需要对结果模1000000007。

示例1:

```
输入：n = 3
输出：4
说明： 有四种走法
```

示例2:

```
输入：n = 5
输出：13
```

提示:

1. n范围在[1, 1000000]之间

递归实现

```java
class Solution {
    private int mod = 1000000007;
    public int waysToStep(int n) {
        if (n == 1) return 1;
        if (n == 2) return 2;
        if (n == 3) return 4;
        return ((waysToStep(n-1) + waysToStep(n-2))%mod + waysToStep(n-3))%mod;
    }
}
```

执行结果： 超出时间限制 显示详情 >

最后执行的输入：

61

时间复杂度：？

空间复杂度：？

等比数列求和公式

$$S_n = n \times a_1 \quad (q = 1)$$

$$S_n = a_1 \cdot \frac{1 - q^n}{1 - q} = \frac{a_1 - a_n \cdot q}{1 - q} \quad (q \neq 1)$$

公式描述： 公式中a1为首项，an为数列第n项，q为等比数列公比，Sn为前n项和。

递归+备忘录

```java
class Solution {
    private int mod = 1000000007;
    private int[] memo = new int[1000001];
    public int waysToStep(int n) {
        if (n == 1) return 1;
        if (n == 2) return 2;
        if (n == 3) return 4;
        if (memo[n] != 0) return memo[n];
        memo[n] = ((waysToStep(n-1) + waysToStep(n-2))%mod + waysToStep(n-3))%mod;
        return memo[n];
    }
}
```

执行结果： 通过 显示详情 >

执行用时： 123 ms ，在所有 Java 提交中击败了 5.07% 的用户

内存消耗： 143.5 MB ，在所有 Java 提交中击败了 5.01% 的用户

时间复杂度：？

空间复杂度：？

等比数列求和公式

$$S_n = n \times a_1 \quad (q = 1)$$

$$S_n = a_1 \cdot \frac{1 - q^n}{1 - q} = \frac{a_1 - a_n \cdot q}{1 - q} \quad (q \neq 1)$$

公式描述： 公式中a1为首项，an为数列第n项，q为等比数列公比，Sn为前n项和。

非递归实现

```java
class Solution {
    public int waysToStep(int n) {
        if (n == 1) return 1;
        if (n == 2) return 2;
        if (n == 3) return 4;
        int[] dp = new int[n+1];
        dp[1] = 1;
        dp[2] = 2;
        dp[3] = 4;
        for (int i = 4; i <= n; i++) {
            dp[i] = ((dp[i-1]+dp[i-2])%1000000007 + dp[i-3])%1000000007;
        }
        return dp[n];
    }
}
```

执行结果： 通过 显示详情 >

执行用时： 39 ms ，在所有 Java 提交中击败了 47.99% 的用户

内存消耗： 42.3 MB ，在所有 Java 提交中击败了 30.44% 的用户

非递归实现+优化

```java
class Solution {
    public int waysToStep(int n) {
        if (n == 1) return 1;
        if (n == 2) return 2;
        if (n == 3) return 4;
        int a = 1;
        int b = 2;
        int c = 4;
        int d = 0;
        for (int i = 4; i <= n; i++) {
            d = ((c+b)%1000000007 + a)%1000000007;
            a = b;
            b = c;
            c = d;
        }
        return d;
    }
}
```

习题课：递归和分治

剑指 Offer 06. 从尾到头打印链表 （简单）**已讲**

输入一个链表的头节点，从尾到头反过来返回每个节点的值（用数组返回）。

示例 1：

```
输入：head = [1,3,2]
输出：[2,3,1]
```

限制：

```
0 <= 链表长度 <= 10000
```

实现方法一：

```
class Solution {
    List<Integer> result = new ArrayList<>();

    public int[] reversePrint(ListNode head) {
        reverseTravel(head);
        int[] resultArr = new int[result.size()];
        int i = 0;
        for (Integer k : result) {
            resultArr[i++] = k;
        }
        return resultArr;
    }

    private void reverseTravel(ListNode head) {
        if (head == null) return;
        reverseTravel(head.next);
        result.add(head.val);
    }
}
```

实现方法二：

```
class Solution {
    public int[] reversePrint(ListNode head) {
        List<Integer> result = new ArrayList<>();
        reverseTravel(head, result);
        int[] resultArr = new int[result.size()];
        int i = 0;
        for (Integer k : result) {
            resultArr[i++] = k;
        }
        return resultArr;
    }

    private void reverseTravel(
            ListNode head, List<Integer> result) {
        if (head == null) return;
        reverseTravel(head.next, result);
        result.add(head.val);
    }
}
```

实现方法三：

```
class Solution {
    public int[] reversePrint(ListNode head) {
        if (head == null) return new int[0];
        int[] subresult = reversePrint(head.next);
        int[] result = new int[subresult.length+1];
        for (int i = 0; i < subresult.length; ++i) {
            result[i] = subresult[i];
        }
        result[result.length-1] = head.val;
        return result;
    }
}
```

习题课：递归和分治

剑指 Offer 24. 反转链表 （简单）

定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点。

示例:

```
输入：1->2->3->4->5->NULL
输出：5->4->3->2->1->NULL
```

限制：

```
0 <= 节点个数 <= 5000
```

```java
class Solution {
    public ListNode reverseList(ListNode head) {
        if (head == null) return null;
        if (head.next == null) return head;   <—递归退出条件
        ListNode newHead = reverseList(head.next);
        head.next.next = head;
        head.next = null;
        return newHead;
    }
}
```

时间复杂度：？
空间复杂度：？

剑指 Offer 25. 合并两个排序的链表 （中等）

输入两个递增排序的链表，合并这两个链表并使新链表中的节点仍然是递增排序的。

示例1：

```
输入：1->2->4, 1->3->4
输出：1->1->2->3->4->4
```

限制：

```
0 <= 链表长度 <= 1000
```

习题课：递归和分治

剑指 Offer 25. 合并两个排序的链表 （中等）

```
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;
        if (l1.val < l2.val) {
            ListNode subHead = mergeTwoLists(l1.next, l2);
            l1.next = subHead;
            return l1;
        } else {
            ListNode subHead = mergeTwoLists(l1, l2.next);
            l2.next = subHead;
            return l2;
        }
    }
}
```

习题课：递归和分治

王争的算法训练营

剑指 Offer 16. 数值的整数次方 （中等）

实现 pow(*x*, *n*) ，即计算 x 的 n 次幂函数（即，x^n）。不得使用库函数，同时不需要考虑大数问题。

示例 1：

```
输入：x = 2.00000, n = 10
输出：1024.00000
```

示例 2：

```
输入：x = 2.10000, n = 3
输出：9.26100
```

示例 3：

```
输入：x = 2.00000, n = -2
输出：0.25000
解释：2^-2 = 1/2^2 = 1/4 = 0.25
```

提示：

- $-100.0 < x < 100.0$
- $-2^{31} <= n <= 2^{31}-1$
- $-10^4 <= x^n <= 10^4$

```java
class Solution {
    public double myPow(double x, int n) {
        if (n >= 0) return rPow(x, n);
        // else return 1 / rPow(x, -1*n);
        // x=2.00000 n=-2147483648
        else return 1/(rPow(x, -1*(n+1))*x);
    }

    public double rPow(double x, int n) {
        if (n == 0) return 1;
        double halfPow = rPow(x, n/2);
        if (n % 2 == 1) {
            return halfPow * halfPow * x;
        } else {
            return halfPow * halfPow;
        }
    }
}
```

时间复杂度：?

空间复杂度：?

习题课：递归和分治

王争的算法训练营

面试题 08.05. 递归乘法 （中等）

递归乘法。 写一个递归函数，不使用 * 运算符， 实现两个正整数的相乘。可以使用加号、减号、位移，但要吝啬一些。

示例1:

```
输入：A = 1, B = 10
输出：10
```

示例2:

```
输入：A = 3, B = 4
输出：12
```

提示:

1. 保证乘法范围不会溢出

```java
class Solution {
    public int multiply(int A, int B) {
        // a个b相加
        if (A == 1) return B;
        int halfValue = multiply(A/2, B);
        if (A%2 == 1) {
            return halfValue+halfValue+B;
        } else {
            return halfValue+halfValue;
        }
    }
}
```

时间复杂度：？

空间复杂度：？

```java
class Solution {
    public int multiply(int A, int B) {
        int n = Math.min(A, B);
        int k = Math.max(A, B);
        if (n == 1) return k;
        // n个k相加=(n/2个k相加)+(n/2个k相加)+0(或k)
        int halfValue = multiply(n/2, k);
        if (n%2 == 1) {
            return halfValue+halfValue+k;
        } else {
            return halfValue+halfValue;
        }
    }
}
```

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料