链滴

Leetcode 每日一题：306. 累加数

作者：Hildaquan
原文链接：https://ld246.com/article/1641788529074
来源网站：链滴

自己尝试

理解错题意了。。。

我以为每一位都可以进行想加，所以就将 string 直接转为 char 数组，然后前后验证一下。结果只通了一半

```
class Solution {
    // 提升作用域，方便dfs遍历
    String num;
    // divided num
    List<List<Integer>> ans = new ArrayList<>();
    int n;  // length
    public boolean isAdditiveNumber(String _num) {
        num = _num;
        n = num.length();
        return dfs(0);
    }

    // divide [startIndex, n - 1], check if it's right
    private boolean dfs(int startIndex) {
        int m = ans.size();
        // if m < 3, it is wrong because it only has two or less nums, which can't get thrid num
        if (startIndex == n) return m >= 3;

        // if num[i] == 0, it can't get a right num, it only can be added as a component
        int endIndex = num.charAt(startIndex) == '0' ? startIndex + 1 : n;

        List<Integer> cur = new ArrayList<>();
        for (int i = startIndex; i < endIndex; i++) {
            // reversedOrdered storing
            cur.add(0, num.charAt(i) - '0');
            // less than 2, add num into ans. Otherwise, check if a + b == cur
            if (m < 2 || check(ans.get(m - 2), ans.get(m - 1), cur)) {
                ans.add(cur);
                if (dfs(i + 1)) return true;
                ans.remove(ans.size() - 1);
            }
        }

        return false;
    }

    private boolean check(List<Integer> a, List<Integer> b, List<Integer> c) {
        List<Integer> sum = new ArrayList<>();
        int t = 0;
        for (int i = 0; i < a.size() || i < b.size(); i++) {
            if (i < a.size()) t += a.get(i);
            if (i < b.size()) t += b.get(i);
            sum.add(t % 10);
            t /= 10;
        }
        // add the last num
```

```
        if (t > 0) sum.add(t);

        // compare to c
        if (sum.size() != c.size()) return false;
        for (int i = 0; i < sum.size(); i++) {
            if (sum.get(i) != c.get(i)) return false;
        }

        return true;
    }
}
```

这么看来的话，这道题就属于难度比较高的 字符串操作 了。。。

宫水三叶的题解

回溯基础

高精度加法

```
class Solution {
    // 提升作用域，方便dfs遍历
    String num;
    // divided num
    List<List<Integer>> ans = new ArrayList<>();
    int n;  // length
    public boolean isAdditiveNumber(String _num) {
        num = _num;
        n = num.length();
        return dfs(0);
    }

    // divide [startIndex, n - 1], check if it's right
    private boolean dfs(int startIndex) {
        int m = ans.size();
        // if m < 3, it is wrong because it only has two or less nums, which can't get thrid num
        if (startIndex == n) return m >= 3;

        // if num[i] == 0, it can't get a right num, it only can be added as a component
        int endIndex = num.charAt(startIndex) == '0' ? startIndex + 1 : n;

        List<Integer> cur = new ArrayList<>();
        for (int i = startIndex; i < endIndex; i++) {
            // reversedOrdered storing
            cur.add(0, num.charAt(i) - '0');
            // less than 2, add num into ans. Otherwise, check if a + b == cur
            if (m < 2 || check(ans.get(m - 2), ans.get(m - 1), cur)) {
                ans.add(cur);
                if (dfs(i + 1)) return true;
                ans.remove(ans.size() - 1);
            }
        }

        return false;
```

```
    }

    private boolean check(List<Integer> a, List<Integer> b, List<Integer> c) {
        List<Integer> sum = new ArrayList<>();
        int t = 0;
        for (int i = 0; i < a.size() || i < b.size(); i++) {
            if (i < a.size()) t += a.get(i);
            if (i < b.size()) t += b.get(i);
            sum.add(t % 10);
            t /= 10;
        }
        // add the last num
        if (t > 0) sum.add(t);

        // compare to c
        if (sum.size() != c.size()) return false;
        for (int i = 0; i < sum.size(); i++) {
            if (sum.get(i) != c.get(i)) return false;
        }

        return true;
    }
}
```

用回溯，一个一个的进行遍历，找到所有可能的结果。

由于存在越界的可能，所以需要高精度加法，所以需要对数据进行倒序存储，倒序相加

从 [u, n - 1] 开始找。u 会递增，说明 u 之前的已经是正确的数值了。

但是这个算法还不是最好的，因为使用了过多的额外空间。在其他题解上还看到使用很少的额外空间可以解出来的。

优化后的回溯

不使用额外的空间存储，直接在每一层中记录，并且在每一层都进行比较。当前层比较失败后，回溯上一层，继续添加上一层的数值，直到该层判断失败或成功。

与三叶的解法最大的不同在于：三叶的解法将每一次成功拆分的数值都存起来。而优化过后的算法，用将每一次都结果存起来，只需要存前三个即可！

```
class Solution {
    public boolean isAdditiveNumber(String num) {
        return dfs(num, 0, 0, 0, 0);
    }

    /*
        count: 代表处理了个数
        prepre：当前数的前两个
        pre：当前数的前一个
    */
    private boolean dfs(String num, int index, int count, long prepre, long pre) {
        // 如果index已经遍历到了末尾，为终止条件
        if (index >= num.length()) return count >= 3;   // 处理个数少于3个，错误
```

```
        long cur = 0;
        for (int i = index; i < num.length(); i++) {
            // 前驱为0，可以作为单个元素。但是不能够出现 0xx 的情况
            if (num.charAt(index) == '0' && i > index) return false;

            // 得到当前的值
            cur = cur * 10 + num.charAt(i) - '0';

            // 小于2的，得继续添加
            if (count >= 2) {
                long sum = prepre + pre;
                // 当前的cur大于sum，false
                if (cur > sum) return false;
                // 当前的cur小于sum，说明还可以继续添加
                if (cur < sum) continue;
            }

            // 没错之后，继续进入下一层
            if (dfs(num, i + 1, count + 1, pre, cur)) return true;
            // 返回错误的话，回溯，继续在这一层继续进行
        }

        return false;
    }
}
```

我认为解法 2 最重要的一步就是 cur < sum 后，不是判断 false ，而是应该继续添加下去。

sum 是上一层已经判断成功的两个数的和，求 cur 就是求第三个数。cur < sum 意味着 cur 的取值未到终止的时候，需要继续在这一层继续取。

跳到下一层的条件只有当这一层的 cur == sum 才可以！

总结

这道题直接考察的是代码的掌控程度，没什么特别复杂的算法，关键在于自己能不能将解题思路用代求出来。

这道题很显然就是想到了暴力遍历，但是暴力遍历也不太简单，要考虑很多附带的条件。

字符怎么拆分

答案怎么存储

三叶姐的算法融合了 回溯算法 和 高精度加法，所以看起来会很复杂。但实际上这是很符合题意的。

高精度加法 要求要逆序相加，所以在算法上，三叶的要复杂很多

优化过后的算法，也是基于三叶姐的改进过来的。三叶姐的算法全局记录拆分的结果，但是实际上只要记录前三个数值就可以了，剩下的数可以递推下去求解正确性