23 括号生成

更新时间：2019-09-05 09:49:33

> 才能一旦让懒惰支配，它就一无可为。
>
> ——克雷洛夫

刷题内容

> 难度：Medium
>
> 原题连接：https://leetcode.com/problems/generate-parentheses/

内容描述

给出 n 代表生成括号的对数，请你写出一个函数，使其能够生成所有可能的并且有效的括号组合。

例如，给出 n = 3，生成结果为：

```
[
  "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
```

解题方案

思路 1：时间复杂度：O(4^N / sqrt(N)) 空间复杂度：O(4^N / sqrt(N))

这道题是一个枚举题，首先我们看一下有效括号的条件：

- 左括号和右括号相等
- 在括号字符串中的任一前缀，右括号的数量不大于左括号的数量

假设我们已经枚举到了某个前缀S1，接下来的字符有两种情况，加上'('或者加上')'，加上之后，如果该前缀不符合有效括号的条件，我们应该放弃这次枚举

比如：前缀'()'，后面一个字符只能是'('

我们用递归的方式来实现这个题目

为了满足有效括号的条件，我们在枚举前缀时，需要左括号的数量和右括号的数量，维护这两个值，有利于我们判断是否满足条件

Python beats 100%

```
class Solution:
    def generateParenthesis(self, n):
        """
        :type n: int
        :rtype: List[str]
        """
        self.res = []
        self.singleStr('', 0, 0, n)
        return self.res

    def singleStr(self, s, left, right, n):
        if left == n and right == n:
            self.res.append(s)
        if left < n:
            self.singleStr(s + '(',left + 1, right, n)
        if right < left:
            self.singleStr(s + ')',left, right + 1, n)
```

Java beats 93.50%

```
class Solution {
    public List<String> generateParenthesis(int n) {
        List<String> result = new ArrayList<>();
        singleStr(result,"",0,0,n);
        return result;
    }

    private void singleStr(List<String> result, String str,int left, int right, int n){
        if (left == n && right == n) {
            result.add(str);
        }
        if (left < n) {
            singleStr(result,str + "(", left + 1,right,n);
        }
        if (right < left) {
            singleStr(result, str + ")", left, right+1,n);
        }
    }
}
```

c++

beats 92.69%

```
class Solution {
public:
    //ret：结果数组，传引用有助于把结果传出来
    //s：当前生成的字符串
    //left：当前左括号的个数
    //right：当前右括号的个数
    //n：要求字符串的左右括号的个数
    void generate(vector<string> & ret, string & s, int left, int right, int n) {
        if (left == n && right == n) {
            ret.push_back(s);
        } else {
            //判断能否加上左括号
            if (left < n) {
                s[left + right] = '(';
                generate(ret, s, left + 1, right, n);
            }
            //判断能否加上右括号
            if (left > right) {
                s[left + right] = ')';
                generate(ret, s, left, right + 1, n);
            }
        }
    }
    vector<string> generateParenthesis(int n) {
        vector<string> ret;
        //这里提前生成2*n个字符的字符串，方便后面的更新
        string s(2 * n, '(');
        generate(ret, s, 0, 0, n);
        return ret;
    }
};
```

go

beats 97.23%

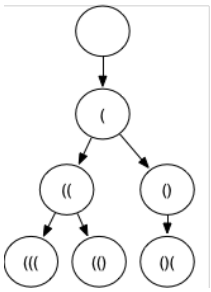
```go
//ret：结果数组，传指针有助于把结果传出来
//s：当前生成的字符串
//left：当前左括号的个数
//right：当前右括号的个数
//n：要求字符串的左右括号的个数
func generate(ret * []string, s []rune, left int, right int, n int) {
  if left == n && right == n {
    tmp := make([]rune, len(s))
       //这里如果不copy，s是个切片，随时会跟着原数据改变
    copy(tmp, s)
    *ret = append(*ret, string(tmp))
    return
  }
    //判断能否加上左括号
  if left < n {
    s[left + right] = '('
    generate(ret, s, left + 1, right, n)
  }
  //判断能否加上右括号
  if left > right {
    s[left + right] = ')'
    generate(ret, s, left, right + 1, n)
  }
}

func generateParenthesis(n int) []string {
  ret := make([]string, 0)
  s := make([]rune, 2 * n)
  generate(&ret, s, 0, 0, n)
  return ret
}
```

小结

递归是个重要的知识点，面试题中经常会出现树的遍历。大家试着把括号生成这个题，跟树的遍历结合起来思考，其实括号生成这个题本质上边递归，边生成出一颗递归树：在每个节点我们计算约束条件，判断是否需要生成出子节点。

递归，本质上是对搜索节点进行无差别遍历。



}

22 删除链表的倒数第 n 个节点

24 两两交换链表中的节点