22. Generate Parentheses

Created	@July 16, 2020 2:02 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/generate-parentheses/
Importance	
Tag	NEET Stack backtrack
Video	

Given `n` pairs of parentheses, write a function to *generate all combinations of well-formed parentheses*.

Example 1:

```
Input: n = 3
Output: ["((()))","(()())","(())()","()(())","()()()"]
```

Example 2:

```
Input: n = 1
Output: ["()"]
```

Constraints:

- `1 <= n <= 8`

Solution

方法一：暴力法

思路

我们可以生成所有 2^{2n} 个 '(' 和 ')' 字符构成的序列，然后我们检查每一个是否有效即可。

算法

为了生成所有序列，我们可以使用递归。长度为 n 的序列就是在长度为 $n-1$ 的序列前加一个 '(' 或 ')'。

为了检查序列是否有效，我们遍历这个序列，并使用一个变量 $balance$ 表示左括号的数量减去右括号的数量。如果在遍历过程中 $balance$ 的值小于零，或者结束时 $balance$ 的值不为零，那么该序列就是无效的，否则它是有效的。

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        def generate(A):
            if len(A) == 2*n:
                if valid(A):
                    ans.append("".join(A))
            else:
                A.append('(')
                generate(A)
                A.pop()
                A.append(')')
                generate(A)
                A.pop()

        def valid(A):
            bal = 0
            for c in A:
                if c == '(': bal += 1
                else: bal -= 1
                if bal < 0: return False
            return bal == 0

        ans = []
        generate([])
        return ans

# 作者：LeetCode-Solution
# 链接：https://leetcode.cn/problems/generate-parentheses/solution/gua-hao-sheng-cheng-by-leetcode-solution/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

复杂度分析

- 时间复杂度：$O(2^{2n}n)$，对于 2^{2n} 个序列中的每一个，我们用于建立和验证该序列的复杂度为 $O(n)$。
- 空间复杂度：$O(n)$，除了答案数组之外，我们所需要的空间取决于递归栈的深度，每一层递归函数需要 $O(1)$ 的空间，最多递归 $2n$ 层，因此空间复杂度为 $O(n)$。

回溯法

方法一还有改进的余地：我们可以只在序列仍然保持有效时才添加 '(' 或 ')'，而不是像 方法一 那样每次添加。我们可以通过跟踪到目前为止放置的左括号和右括号的数目来做到这一点，

如果左括号数量不大于 n，我们可以放一个左括号。如果右括号数量小于左括号的数量，我们可以放一个右括号。

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        stack = []
        res = []

        def backtrack(openN, closedN):
            if openN == closedN == n:
                res.append(''.join(stack))
                return

            if openN < n:
                stack.append('(')
                backtrack(openN + 1, closedN)
                stack.pop()

            if closedN < openN:
                stack.append(')')
                backtrack(openN, closedN + 1)
                stack.pop()

        backtrack(0, 0)
        return res
```

我们的复杂度分析依赖于理解 $generateParenthesis(n)$ 中有多少个元素。这个分析超出了本文的范畴，但事实证明这是第 n 个卡特兰数 $\frac{1}{n+1}\binom{2n}{n}$，这是由 $\frac{4^n}{n\sqrt{n}}$ 渐近界定的。

- 时间复杂度：$O(\frac{4^n}{\sqrt{n}})$，在回溯过程中，每个答案需要 $O(n)$ 的时间复制到答案数组中。
- 空间复杂度：$O(n)$，除了答案数组之外，我们所需要的空间取决于递归栈的深度，每一层递归函数需要 $O(1)$ 的空间，最多递归 $2n$ 层，因此空间复杂度为 $O(n)$。

力扣

https://leetcode.cn/problems/generate-parentheses/solution/gua-hao-sheng-cheng-by-leetcode-solution/