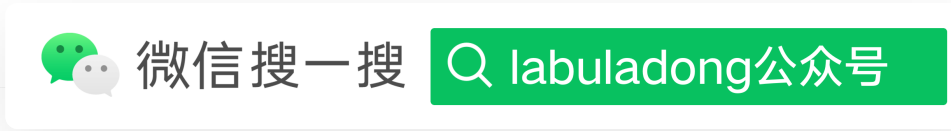
回溯算法最佳实践：括号生成



通知： **数据结构精品课 V1.7 持续更新中；第九期打卡挑战 开始报名；B 站可查看核心算法框架系列视频。**

读完本文，你不仅学会了算法套路，还可以顺便解决如下题目：

牛客	LeetCode	力扣	难度
-	22. Generate Parentheses	22. 括号生成	🟠
-	-	剑指 Offer II 085. 生成匹配的括号	🟠

——————

括号问题可以简单分成两类，一类是前文写过的 括号的合法性判断 ，一类是合法括号的生成。对于括号合法性的判断，主要是借助「栈」这种数据结构，而对于括号的生成，一般都要利用回溯递归的思想。

关于回溯算法，我们前文写过一篇 回溯算法套路框架详解 反响非常好，读本文前应该读过那篇文章，这样你就能够进一步了解回溯算法的框架使用方法了。

回到正题，看下力扣第 22 题「 括号生成」，要求如下：

请你写一个算法，输入是一个正整数 `n`，输出是 `n` 对儿括号的所有合法组合，函数签名如下：

```
vector<string> generateParenthesis(int n);
```

比如说，输入 `n=3`，输出为如下 5 个字符串：

```
"((()))",
"(()())",
"(())()",
"()(())",
"()()()"
```

有关括号问题，你只要记住以下性质，思路就很容易想出来：

1、一个「合法」括号组合的左括号数量一定等于右括号数量，这个很好理解。

2、对于一个「合法」的括号字符串组合 `p`，必然对于任何 `0 <= i < len(p)` 都有：子串 `p[0..i]` 中左括号的数量都大于或等于右括号的数量。

如果不跟你说这个性质，可能不太容易发现，但是稍微想一下，其实很容易理解，因为从左往右算的话，肯定是左括号多嘛，到最后左右括号数量相等，说明这个括号组合是合法的。

反之，比如这个括号组合 `))((`，前几个子串都是右括号多于左括号，显然不是合法的括号组合。

下面就来手把手实践一下回溯算法框架。

回溯思路

明白了合法括号的性质，如何把这道题和回溯算法扯上关系呢？

算法输入一个整数 `n`，让你计算 `n` **对儿括号**能组成几种合法的括号组合，可以改写成如下问题：

现在有 `2n` 个位置，每个位置可以放置字符 `(` 或者 `)`，组成的所有括号组合中，有多少个是合法的？

这个命题和题目的意思完全是一样的对吧，那么我们先想想如何得到全部 `2^(2n)` 种组合，然后再根据我们刚才总结出的合法括号组合的性质筛选出合法的组合，不就完事儿了？

如何得到所有的组合呢？这就是标准的暴力穷举回溯框架啊，我们前文 回溯算法套路框架详解 都总结过了：

```
result = []
def backtrack(路径, 选择列表):
    if 满足结束条件:
        result.add(路径)
        return

    for 选择 in 选择列表:
        做选择
        backtrack(路径, 选择列表)
        撤销选择
```

那么对于我们的需求，如何打印所有括号组合呢？套一下框架就出来了，伪码如下：

```
void backtrack(int n, int i, string& track) {
    // i 代表当前的位置，共 2n 个位置
    // 穷举到最后一个位置了，得到一个长度为 2n 组合
    if (i == 2 * n) {
        print(track);
        return;
    }

    // 对于每个位置可以是左括号或者右括号两种选择
    for choice in ['(', ')'] {
        track.push(choice); // 做选择
        // 穷举下一个位置
        backtrack(n, i + 1, track);
        track.pop(choice); // 撤销选择
    }
}
```

那么，现在能够打印所有括号组合了，如何从它们中筛选出合法的括号组合呢？很简单，加几个条件进行「剪枝」就行了。

对于 `2n` 个位置，必然有 `n` 个左括号，`n` 个右括号，所以我们不是简单的记录穷举位置 `i`，而是**用 `left` 记录还可以使用多少个左括号，用 `right` 记录还可以使用多少个右括号**，这样就可以通过刚才总结的合法括号规律进行筛选了：

```
vector<string> generateParenthesis(int n) {
    if (n == 0) return {};
    // 记录所有合法的括号组合
    vector<string> res;
    // 回溯过程中的路径
    string track;
    // 可用的左括号和右括号数量初始化为 n
    backtrack(n, n, track, res);
    return res;
}

// 可用的左括号数量为 left 个，可用的右括号数量为 rgiht 个
void backtrack(int left, int right,
               string& track, vector<string>& res) {
    // 若左括号剩下的多，说明不合法
    if (right < left) return;
    // 数量小于 0 肯定是不合法的
    if (left < 0 || right < 0) return;
    // 当所有括号都恰好用完时，得到一个合法的括号组合
    if (left == 0 && right == 0) {
        res.push_back(track);
        return;
    }

    // 尝试放一个左括号
    track.push_back('('); // 选择
    backtrack(left - 1, right, track, res);
    track.pop_back(); // 撤消选择

    // 尝试放一个右括号
    track.push_back(')'); // 选择
    backtrack(left, right - 1, track, res);
    track.pop_back(); // 撤消选择
}
```

这样，我们的算法就完成了，算法的复杂度是多少呢？这个比较难分析，**对于递归相关的算法，时间复杂度这样计算（递归次数）*（递归函数本身的时间复杂度）**。

`backtrack` 就是我们的递归函数，其中没有任何 for 循环代码，所以递归函数本身的时间复杂度是 O(1)，但关键是这个函数的递归次数是多少？换句话说，给定一个 `n`，`backtrack` 函数递归被调用了多少次？

我们前面怎么分析动态规划算法的递归次数的？主要是看「状态」的个数对吧。其实回溯算法和动

态规划的本质都是穷举，只不过动态规划存在「重叠子问题」可以优化，而回溯算法不存在而已。

所以说这里也可以用「状态」这个概念，**对于 `backtrack` 函数，状态有三个，分别是 `left, right, track`**，这三个变量的所有组合个数就是 `backtrack` 函数的状态个数（调用次数）。

`left` 和 `right` 的组合好办，他俩取值就是 0~n 嘛，组合起来也就 `n^2` 种而已；这个 `track` 的长度虽然取在 0~2n，但对于每一个长度，它还有指数级的括号组合，这个是不好算的。

说了这么多，就是想让大家知道这个算法的复杂度是指数级，而且不好算，这里就不具体展开了，是 `4^n / sqrt(n)`，有兴趣的读者可以搜索一下「卡特兰数」相关的知识了解一下这个复杂度是怎么算的。

▸ **引用本文的题目**

▸ **引用本文的文章**

- - - - - - - - - - - - - -

《labuladong 的算法小抄》已经出版，关注公众号查看详情；后台回复关键词「进群」可加入算法群；回复「PDF」可获取精华文章 PDF：

共同维护高质量学习环境，评论礼仪见这里，违者直接拉黑不解释