DFS：用递归、栈

recursion

```
visited = set()

def dfs(node, visited):
    if node in visited:    # terminator
        # already visited
        return

    visited.add(node)

    # process current node here.
    ...
    for next_node in node.children():
        if next_node not in visited:
            dfs(next_node, visited)
```

stack

```
def DFS(self, tree):
    if tree.root is None:
        return []

    visited, stack = [], [tree.root]

    while stack:
        node = stack.pop()
        visited.add(node)

        process(node)
        nodes = generate_related_nodes(node)
        stack.push(nodes)

    # other processing work
    ...
```

BFS：用队列

```
# BFS
def BFS(graph, start, end):
    visited = set()
    queue = []
    queue.append([start])

    while queue:
        node = queue.pop()
        visited.add(node)

        processs(node)
        nodes = generate_related_nodes(node)
```

```
            queue.push(nodes)

        # other processing word
        ...
```

102. 二叉树的层序遍历

(https://leetcode-cn.com/problems/binary-tree-level-order-traversal/)

难度中等556

给你一个二叉树，请你返回其按 **层序遍历** 得到的节点值。 （即逐层地，从左到右访问所有节点）。

示例：
二叉树：`[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

返回其层次遍历结果：

```
[
  [3],
  [9,20],
  [15,7]
]
```

DFS分析

代码

Python by 负雪明烛

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        # 32ms, beats 98%
        res = []
        self.level(root, 0, res)
        return res

    def level(self, root, level, res):
        if not root: return
        if len(res) == level: res.append([])
        res[level].append(root.val)
```

```
        if root.left: self.level(root.left, level + 1, res)
        if root.right: self.level(root.right, level + 1, res)
```

stack

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
# 0503 11:41
class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root:return []
        #init
        stack = [root]
        subStack = []
        #return
        res = []

        while stack or subStack:
            tmp = []
            while stack:
                node = stack.pop()
                tmp.append(node.val)
                if node.left: subStack.append(node.left)
                if node.right: subStack.append(node.right)

            res.append(tmp)
            stack = subStack[::-1]
            subStack = []

        return res
# 0503 11:48
```

22. 括号生成

难度中等1135

数字 *n* 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且 **有效的** 括号组合。

示例：

```
输入：n = 3
输出：[
       "((()))",
       "(()())",
       "(())()",
       "()(())",
       "()()()"
     ]
```

Clean Python DP Solution ***

To generate all n-pair parentheses, we can do the following:

Generate one pair: ()

Generate 0 pair inside, n - 1 afterward: () (...)...

Generate 1 pair inside, n - 2 afterward: (()) (...)...

...

Generate n - 1 pair inside, 0 afterward: ((...))

I bet you see the overlapping subproblems here. Here is the code:

(you could see in the code that x represents one j-pair solution and y represents one (i - j - 1) pair solution, and we are taking into account all possible of combinations of them)

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        # time: 24ms,  beats 99%
        dp = [[] for i in range(n+1)]
        dp[0].append('')
        for i in range(n + 1):
            for j in range(i):
                dp[i] += ['(' + x + ')' + y for x in dp[j] for y in dp[i - j -
1]]
        return dp[n]
```

Simple Python DFS solution with explanation

If you have two stacks, one for n "(", the other for n ")", you generate a binary tree from these two stacks of left/right parentheses to form an output string.

This means that whenever you traverse deeper, you pop one parentheses from one of stacks. When two stacks are empty, you form an output string.

How to form a legal string? Here is the simple observation:

- For the output string to be right, stack of ")" most be larger than stack of "(". If not, it creates string like "())"
- Since elements in each of stack are the same, we can simply express them with a number. For example, left = 3 is like a stacks ["(", "(", "("]

So, here is my sample code in Python:

```
class Solution:
    def generateParenthesis(self, n):
        if not n:
            return []
        left, right, ans = n, n, []
        self.dfs(left,right, ans, "")
        return ans

    def dfs(self, left, right, ans, string):
        if right < left:
            return
        if not left and not right:
            ans.append(string)
```

```
            return
        if left:
            self.dfs(left-1, right, ans, string + "(")
        if right:
            self.dfs(left, right-1, ans, string + ")")
```

4-7 lines Python StefanPochmann

`p` is the parenthesis-string built so far, `left` and `right` tell the number of left and right parentheses still to add, and `parens` collects the parentheses.

Solution 1

I used a few "tricks"... how many can you find? :-)

```
def generateParenthesis(self, n):
    def generate(p, left, right, parens=[]):
        if left:         generate(p + '(', left-1, right)
        if right > left: generate(p + ')', left, right-1)
        if not right:    parens += p,
        return parens
    return generate('', n, n)
```

Solution 2

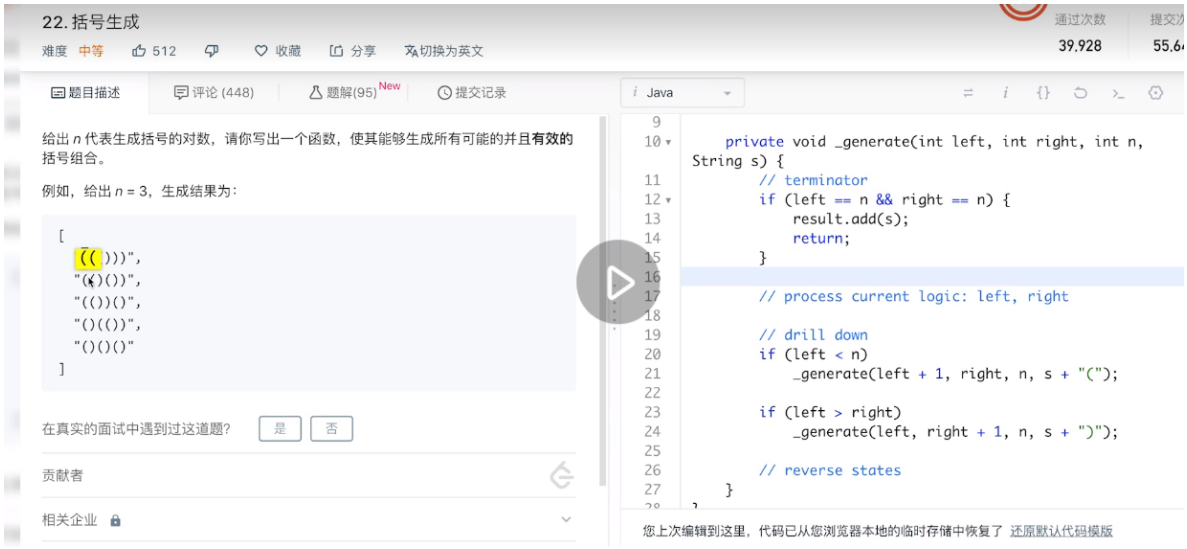
Here I wrote an actual Python generator. I allow myself to put the `yield q` at the end of the line because it's not that bad and because in "real life" I use Python 3 where I just say `yield from generate(...)`.

```
def generateParenthesis(self, n):
    def generate(p, left, right):
        if right >= left >= 0:
            if not right:
                yield p
            for q in generate(p + '(', left-1, right): yield q
            for q in generate(p + ')', left, right-1): yield q
    return list(generate('', n, n))
```

Solution 3

Improved version of this. Parameter `open` tells the number of "already opened" parentheses, and I continue the recursion as long as I still have to open parentheses (`n > 0`) and I haven't made a mistake yet (`open >= 0`).

```
def generateParenthesis(self, n, open=0):
    if n > 0 <= open:
        return ['(' + p for p in self.generateParenthesis(n-1, open+1)] + \
               [')' + p for p in self.generateParenthesis(n, open-1)]
    return [')' * open] * (not n)
```


22. 括号生成
难度 中等
512
收藏
分享
切换为英文
通过次数
39,928
题目描述
评论 (448)
题解(95) New
提交记录
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：
[
"((()))",
"(()())",
"(())()",
"()(())",
"()()()"
]
在真实的面试中遇到过这道题?
是
否
贡献者
相关企业
Java
private void _generate(int left, int right, int n, String s) {
// terminator
if (left == n && right == n) {
result.add(s);
return;
}
// process current logic: left, right
// drill down
if (left < n)
_generate(left + 1, right, n, s + "(");
if (left > right)
_generate(left, right + 1, n, s + ")");
// reverse states
}
您上次编辑到这里，代码已从您浏览器本地的临时存储中恢复了 还原默认代码模版