22 Generate Parentheses

Problem Description

The problem requires us to generate all possible combinations of well-formed parentheses given n pairs. A well-formed combination of parentheses means that each opening bracket "(" has a corresponding closing bracket ")", and they are correctly nested. To better understand, for n=3, one such correct combination would be "((()))", whereas "(()" or "())(" would be incorrect formations.

Intuition

To arrive at the solution, we need to think about how we can ensure we create well-formed parentheses. For that, we use Depth First Search (DFS), which is a recursive method to explore all possible combinations of the parentheses.

1. We start with an empty string and at each level of the recursion we have two choices: add an opening parenthesis "(" or a closing parenthesis ")".
2. However, we have to maintain the correctness of the parentheses. This means we cannot add a closing parenthesis if there are not enough opening ones that need closing.
3. We keep track of the number of opening and closing parentheses used so far. We are allowed to add an opening parenthesis if we have not used all `n` available opening parentheses.
4. We can add a closing parenthesis if the number of closing parentheses is less than the number of opening parentheses used. This ensures we never have an unmatched closing parenthesis.
5. We continue this process until we have used all `n` pairs of parentheses.
6. When both the opening and closing parentheses counts equal `n`, it means we have a valid combination, so we add it to our list of answers.

The code uses a helper function `dfs` which takes 3 parameters: the number of opening and closing parentheses used so far (`l` and `r`), and the current combination of parentheses (`t`).

Complexity

Time Complexity

The time complexity of the given code is `O(4^n / sqrt(n))`. This complexity arises because each valid combination can be represented by a path in a decision tree, which has `2n` levels (since we make a decision at each level to add either a left or a right parenthesis, and we do this `n` times for each parenthesis type). However, not all paths in the tree are valid; the number of valid paths follows the nth Catalan number, which is proportional to `4^n / (n * sqrt(n))`, and `n` is a factor that represents the polynomial part that gets smaller as `n` gets larger. Since we're looking at big-O notation, we simplify this to `4^n / sqrt(n)` for large `n`.

Space Complexity

The space complexity is `O(n)` because the depth of the recursive call stack is proportional to the number of parentheses to generate, which is `2n`, and the space required to store a single generated set of parentheses is also linear to `n`.