Generate Parentheses

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

For example, given *n* = 3, a solution set is:

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

Solution 1

The idea is intuitive. Use two integers to count the remaining left parenthesis (n) and the right parenthesis (m) to be added. At each function call add a left parenthesis if n >0 and add a right parenthesis if m>0. Append the result and terminate recursive calls when both m and n are zero.

```
class Solution {
public:
    vector<string> generateParenthesis(int n) {
        vector<string> res;
        addingpar(res, "", n, 0);
        return res;
    }
    void addingpar(vector<string> &v, string str, int n, int m){
        if(n==0 && m==0) {
            v.push_back(str);
            return;
        }
        if(m > 0){ addingpar(v, str+")", n, m-1); }
        if(n > 0){ addingpar(v, str+"(", n-1, m+1); }
    }
};
```

written by klycok original link here

Solution 2

My method is DP. First consider how to get the result f(n) from previous result f(0)...f(n-1). Actually, the result f(n) will be put an extra () pair to f(n-1). Let the "(" always at the first position, to produce a valid result, we can only put ")" in a way that there will be i pairs () inside the extra () and n - 1 - i pairs () outside the extra pair.

Let us consider an example to get clear view:

f(0): ""

f(1): "("f(0)")"

f(2): "("f(0)")"f(1), "("f(1)")"

f(3): "("f(0)")"f(2), "("f(1)")"f(1), "("f(2)")"

So f(n) = "("f(0)")"f(n-1) , "("f(1)")"f(n-2) "("f(2)")"f(n-3) ... "("f(i)")"f(n-1-i) ... "(f(n-1)")"

Below is my code:

```
public class Solution
{
    public List<String> generateParenthesis(int n)
    {
        List<List<String>> lists = new ArrayList<>();
        lists.add(Collections.singletonList(""));

        for (int i = 1; i <= n; ++i)
        {
            final List<String> list = new ArrayList<>();

            for (int j = 0; j < i; ++j)
            {
                for (final String first : lists.get(j))
                {
                    for (final String second : lists.get(i - 1 - j))
                    {
                        list.add("(" + first + ")" + second);
                    }
                }
            }

            lists.add(list);
        }

        return lists.get(lists.size() - 1);
    }
}
```

written by left.peter original link here

Solution 3

```
 public List<String> generateParenthesis(int n) {
        List<String> list = new ArrayList<String>();
        backtrack(list, "", 0, 0, n);
        return list;
    }

    public void backtrack(List<String> list, String str, int open, int close, int
max){

        if(str.length() == max*2){
            list.add(str);
            return;
        }

        if(open < max)
            backtrack(list, str+"(", open+1, close, max);
        if(close < open)
            backtrack(list, str+")", open, close+1, max);
    }
```

The idea here is to only add '(' and ')' that we know will guarantee us a solution (instead of adding 1 too many close). Once we add a '(' we will then discard it and try a ')' which can only close a valid '('. Each of these steps are recursively called.

written by brobins9 original link here