

Grokking the Coding Interview: Patterns for Coding Questions

61% completed



- Search Course
- Reverse every K-element Sub-list (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Breadth First Search

- Introduction
- Binary Tree Level Order Traversal (easy)
- Reverse Level Order Traversal (easy)
- Zigzag Traversal (medium)
- Level Averages in a Binary Tree (easy)
- Minimum Depth of a Binary Tree (easy)
- Level Order Successor (easy)
- Connect Level Order Siblings (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Depth First Search

- Introduction
- Binary Tree Path Sum (easy)
- All Paths for a Sum (medium)
- Sum of Path Numbers (medium)
- Path With Given Sequence (medium)
- Count Paths for a Sum (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Two Heaps

- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern: Subsets

- Introduction
- Subsets (easy)
- Subsets With Duplicates (easy)

Balanced Parentheses (hard)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity
 - Recursive Solution

Problem Statement

For a given number 'N', write a function to generate all combination of 'N' pairs of balanced parentheses.

Example 1:

```
Input: N=2
Output: (()), ()()
```

Example 2:

```
Input: N=3
Output: ((()), ((())), ((())(), ()()), ()())
```

Try it yourself

Try solving this question here:

Java
Python3
JS
C++

```

1 const generate_valid_parentheses = function(num) {
2   result = [];
3   // TODO: Write your code here
4   return result;
5 };
6
7
8 console.log(`All combinations of balanced parentheses are: ${generate_valid_parentheses(2)}`);
9 console.log(`All combinations of balanced parentheses are: ${generate_valid_parentheses(3)}`);

```

RUN
SAVE
RESET


Solution

This problem follows the [Subsets](#) pattern and can be mapped to [Permutations](#). We can follow a similar BFS approach.

Let's take Example-2 mentioned above to generate all the combinations of balanced parentheses. Following a BFS approach, we will keep adding open parentheses `(` or close parentheses `)`. At each step we need to keep two things in mind:

- We can't add more than 'N' open parenthesis.
- To keep the parentheses balanced, we can add a close parenthesis `)` only when we have already added enough open parenthesis `(`. For this, we can keep a count of open and close parenthesis with every combination.

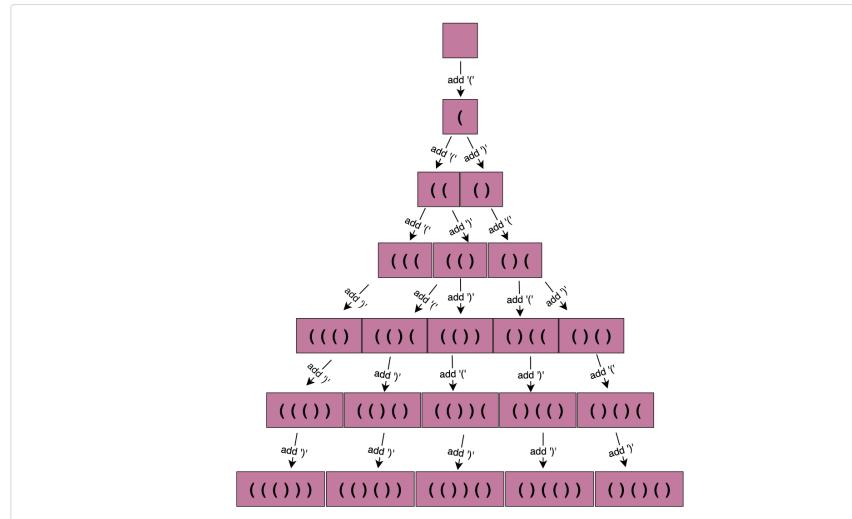
Following this guideline, let's generate parentheses for N=3:

1. Start with an empty combination: `" "`
2. At every step, let's take all combinations of the previous step and add `(` or `)` keeping the above-mentioned two rules in mind.
3. For the empty combination, we can add `(` since the count of open parenthesis will be less than 'N'. We can't add `)` as we don't have an equivalent open parenthesis, so our list of combinations will now be: `" "`
4. For the next iteration, let's take all combinations of the previous set. For `" "` we can add another `(` to it since the count of open parenthesis will be less than 'N'. We can also add `)` as we do have an equivalent open parenthesis, so our list of combinations will be: `"(", ")"`
5. In the next iteration, for the first combination `"("`, we can add another `(` to it as the count of open parenthesis will be less than 'N', we can also add `)` as we do have an equivalent open parenthesis. This gives us two new combinations: `"((" and "()"`. For the second combination `")"`, we can add another `(` to it since the count of open parenthesis will be less than 'N'. We can't add `)` as we don't have an equivalent open parenthesis, so our list of combinations will be: `"((", "(), "())"`
6. Following the same approach, next we will get the following list of combinations: `"((", "(), "())", "(()", "(())"`

	Permutations (medium)
	String Permutations by changing case (medium)
●	Balanced Parentheses (hard)
○	Unique Generalized Abbreviations (hard)
○	Problem Challenge 1
○	Solution Review: Problem Challenge 1
○	Problem Challenge 2
○	Solution Review: Problem Challenge 2
○	Problem Challenge 3
○	Solution Review: Problem Challenge 3
Pattern: Modified Binary Search	^
	Introduction
	Order-agnostic Binary Search (easy)
●	Ceiling of a Number (medium)
○	Next Letter (medium)
○	Number Range (medium)
○	Search in a Sorted Infinite Array (medium)
○	Minimum Difference Element (medium)
○	Bitonic Array Maximum (easy)
○	Problem Challenge 1
○	Solution Review: Problem Challenge 1
○	Problem Challenge 2
○	Solution Review: Problem Challenge 2
○	Problem Challenge 3
○	Solution Review: Problem Challenge 3
Pattern: Bitwise XOR	^
○	Introduction
○	Single Number (easy)
○	Two Single Numbers (medium)
○	Complement of Base 10 Number (medium)
○	Problem Challenge 1
○	Solution Review: Problem Challenge 1
Pattern: Top 'K' Elements	^
	Introduction
●	Top 'K' Numbers (easy)
○	Kth Smallest Number (easy)
○	'K' Closest Points to the Origin (easy)
○	Connect Ropes (easy)
○	Top 'K' Frequent Numbers (medium)
○	Frequency Sort (medium)
○	Kth Largest Number in a Stream (medium)
○	'K' Closest Numbers (medium)
○	Maximum Distinct Elements (medium)
○	Sum of Elements (medium)
○	Rearrange String (hard)
○	Problem Challenge 1
○	Solution Review: Problem Challenge 1
○	Problem Challenge 2
○	Solution Review: Problem Challenge 2
○	Problem Challenge 3
○	Solution Review: Problem Challenge 3

- "00"
7. Next we will get: "((0))", "(00)", "(0)0", "0(0)", "000"
8. Finally, we will have the following combinations of balanced parentheses: "((0))", "(00)", "(0)0", "0(0)", "000"
9. We can't add more parentheses to any of the combinations, so we stop here.

Here is the visual representation of this algorithm:



Code

Here is what our algorithm will look like:

Java Python C++ JS

```

1 const Deque = require('./collections/deque'); //http://www.collectionsjs.com
2
3 class ParenthesesString {
4   constructor(str, openCount, closeCount) {
5     this.str = str;
6     this.openCount = openCount;
7     this.closeCount = closeCount;
8   }
9 }
10
11
12 function generate_valid_parentheses(num) {
13   let result = [],
14   queue = new Deque();
15   queue.push(new ParenthesesString('', 0, 0));
16   while (queue.length > 0) {
17     const ps = queue.shift();
18     // if we've reached the maximum number of open and close parentheses, add to the result
19     if (ps.openCount === num && ps.closeCount === num) {
20       result.push(ps.str);
21     } else {
22       if (ps.openCount < num) { // if we can add an open parentheses, add it
23         queue.push(new ParenthesesString(`${ps.str}(`, ps.openCount + 1, ps.closeCount));
24       }
25       if (ps.openCount > ps.closeCount) { // if we can add a close parentheses, add it
26         queue.push(new ParenthesesString(`${ps.str})`, ps.openCount, ps.closeCount + 1));
27     }
28   }

```

RUN SAVE RESET CLOSE

Output 6.785s

```
All combinations of balanced parentheses are: ((())),(())()
All combinations of balanced parentheses are: ((()),(())(),(())(),(())(),(())())

```

Time complexity

Let's try to estimate how many combinations we can have for 'N' pairs of balanced parentheses. If we don't care for the ordering - *that () can only come after ()* - then we have two options for every position, i.e., either put open parentheses or close parentheses. This means we can have a maximum of 2^N combinations. Because of the ordering, the actual number will be less than 2^N .

If you see the visual representation of Example-2 closely you will realize that, in the worst case, it is equivalent to a binary tree, where each node will have two children. This means that we will have 2^N leaf nodes and $2^N - 1$ intermediate nodes. So the total number of elements pushed to the queue will be $2^N + 2^N - 1$, which is asymptotically equivalent to $O(2^N)$. While processing each element, we do need to concatenate the current string with (or). This operation will take $O(N)$, so the overall time complexity of our algorithm will be $O(N * 2^N)$. This is not completely accurate but reasonable enough to be presented in the interview.

Pattern: K-way merge ^

- Introduction
- Merge K Sorted Lists (medium)
- Kth Smallest Number in M Sorted Lists (Medium)
- Kth Smallest Number in a Sorted Matrix (Hard)
- Smallest Number Range (Hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern : 0/1 Knapsack ^ (Dynamic Programming)

- Introduction
- 0/1 Knapsack (medium)
- Equal Subset Sum Partition (medium)
- Subset Sum (medium)
- Minimum Subset Sum Difference (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2



Explore



Tracks

My Courses



Edpresso



Refer a Friend

Pattern: Topological Sort (Graph) ^

- Introduction
- Topological Sort (medium)
- Tasks Scheduling (medium)
- Tasks Scheduling Order (medium)
- All Tasks Scheduling Orders (hard)
- Alien Dictionary (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Miscellaneous ^

- Kth Smallest Number (hard)

Conclusions ^

- Where to Go from Here



Create

Mark Course as Completed

The actual time complexity ($O(4^n/\sqrt{n})$) is bounded by the [Catalan number](#) and is beyond the scope of a coding interview. See more details [here](#).

Space complexity ^

All the additional space used by our algorithm is for the output list. Since we can't have more than $O(2^N)$ combinations, the space complexity of our algorithm is $O(N * 2^N)$.

Recursive Solution ^

Here is the recursive algorithm following a similar approach:

Java
Python3
C++
JS

```

1 function generate_valid_parentheses(num) {
2     result = [];
3     const parenthesesString = Array(2 * num);
4     generate_valid_parentheses_rec(num, 0, 0, parenthesesString, 0, result);
5     return result;
6 }
7
8
9 function generate_valid_parentheses_rec(num, openCount, closeCount, parenthesesString, index, result) {
10    // if we've reached the maximum number of open and close parentheses, add to the result
11    if (openCount === num && closeCount === num) {
12        result.push(parenthesesString.join(''));
13    } else {
14        if (openCount < num) { // if we can add an open parentheses, add it
15            parenthesesString[index] = '(';
16            generate_valid_parentheses_rec(num, openCount + 1, closeCount, parenthesesString, index + 1, result);
17        }
18        if (openCount > closeCount) { // if we can add a close parentheses, add it
19            parenthesesString[index] = ')';
20            generate_valid_parentheses_rec(num, openCount, closeCount + 1, parenthesesString, index + 1, result);
21        }
22    }
23}
24
25 console.log(`All combinations of balanced parentheses are: ${generate_valid_parentheses(2)}'`);
26 console.log(`All combinations of balanced parentheses are: ${generate_valid_parentheses(3)}'`);
```

RUN
SAVE
RESET

Output
2.090s

All combinations of balanced parentheses are: ()(),()
All combinations of balanced parentheses are: ((())),(()()),((())(),()()),()())

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. [See how](#)

← Back**Next →**

String Permutations by changing case...

Unique Generalized Abbreviations (ha...)

 Mark as Completed

Report an Issue Ask a Question