

Grokking the Coding Interview: Patterns for Coding Questions

62% completed



-  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)
-  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)

Unique Generalized Abbreviations (hard)

We'll cover the following ^

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

Problem Statement

Given a word, write a function to generate all of its unique generalized abbreviations.

Generalized abbreviation of a word can be generated by replacing each substring of the word by the count of characters in the substring. Take the example of "ab" which has four substrings: "", "a", "b", and "ab". After replacing these substrings in the actual word by the count of characters we get all the generalized abbreviations: "ab", "1b", "a1", and "2".

Example 1:

```
Input: "BAT"
Output: "BAT", "BA1", "B1T", "B2", "1AT", "1A1", "2T", "3"
```

Example 2:

```
Input: "code"
Output: "code", "cod1", "cole", "co2", "c1de", "c1d1", "c2e", "c3", "1ode", "1od1", "1o1e", "1o
2",
"2de", "2d1", "3e", "4"
```

Try it yourself

Try solving this question here:

 Java
 Python3
 JS
 C++

```

1 const generate_generalizedAbbreviation = function(word) {
2   result = [];
3   // TODO: Write your code here
4   return result;
5 };
6
7
8 console.log(`Generalized abbreviation are: ${generate_generalizedAbbreviation("BAT")}`)
9 console.log(`Generalized abbreviation are: ${generate_generalizedAbbreviation("code")}`)
10

```

RUN
SAVE
RESET


Solution

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

Let's take Example-1 mentioned above to generate all unique generalized abbreviations. Following a BFS approach, we will abbreviate one character at a time. At each step we have two options:

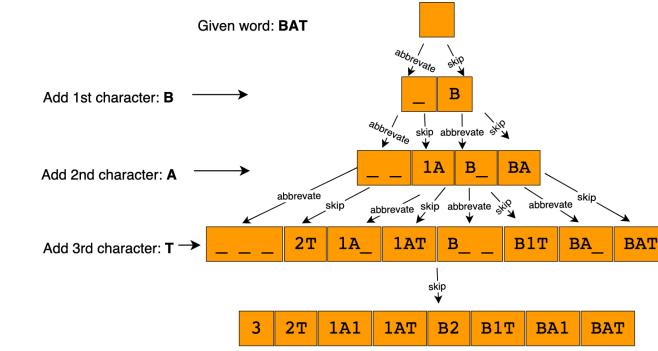
- Abbreviate the current character, or
- Add the current character to the output and skip abbreviation.

Following these two rules, let's abbreviate **BAT**:

1. Start with an empty word: ""
2. At every step, we will take all the combinations from the previous step and apply the two abbreviation rules to the next character.
3. Take the empty word from the previous step and add the first character to it. We can either abbreviate the character or add it (by skipping abbreviation). This gives us two new words: **_**, **B**.
4. In the next iteration, let's add the second character. Applying the two rules on **_** will give us **_** and **1A**. Applying the above rules to the other combination **B** gives us **B_** and **BA**.
5. The next iteration will give us: **_**, **2T**, **1A**, **1AT**, **B**, **1B**, **BA**, **BAT**
6. The final iteration will give us: **3**, **2T**, **1A1**, **1AT**, **B2**, **B1T**, **BA1**, **BAT**

Here is the visual representation of this algorithm:

- Given word: BAT
- 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

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

Code

Here is what our algorithm will look like:

Java
Python3
C++
JS

```

1 const Deque = require('./collections/deque'); //http://www.collectionsjs.com
2
3 class AbbreviatedWord {
4   constructor(str, start, count) {
5     this.str = str;
6     this.start = start;
7     this.count = count;
8   }
9 }
10
11
12 function generate_generalized_abbreviation(word) {
13   let wordlen = word.length,
14     result = [];
15   const queue = new Deque();
16   queue.push(new AbbreviatedWord("", 0, 0));
17   while (queue.length > 0) {
18     const abWord = queue.shift();
19     if (abWord.start === wordlen) {
20       if (abWord.count !== 0) {
21         abWord.str += abWord.count;
22       }
23       result.push(abWord.str);
24     } else {
25       // continue abbreviating by incrementing the current abbreviation count
26       queue.push(new AbbreviatedWord(abWord.str, abWord.start + 1, abWord.count + 1));
27     }
28   }
29   // restart abbreviating, append the count and the current character to the string

```

RUN
SAVE
RESET

Close

Output
4.317s

```
Generalized abbreviation are: 3,2T,1A1,1AT,B2,B1T,BA1,BAT
Generalized abbreviation are: 4,3e,2d1,2de,1o2,1o1e,1od1,1ode,c3,c2e,cld1,c1de,co2,cole,co1,code
```

Time complexity

Since we had two options for each character, we will have a maximum of 2^N combinations. If you see the visual representation of Example-1 closely you will realize that it is equivalent to a binary tree, where each node has 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 a character. This operation will take $O(N)$, so the overall time complexity of our algorithm will be $O(N * 2^N)$.

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_generalized_abbreviation(word) {
2   const result = [];
3   generate_abbreviation_recursive(word, "", 0, 0, result);
4   return result;
5 }
6
7
8 function generate_abbreviation_recursive(word, abWord, start, count, result) {
9   if (start === word.length) {
10     if (count !== 0) {
11       abWord += count;
12     }

```

- (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

```

13     result.push(abWord);
14 } else {
15     // continue abbreviating by incrementing the current abbreviation count
16     generate_abbreviation_recursive(word, abWord, start + 1, count + 1, result);
17
18     // restart abbreviating, append the count and the current character to the string
19     if (count !== 0) {
20         abWord += count;
21     }
22     const newWord = abWord + word[start];
23     generate_abbreviation_recursive(word, newWord, start + 1, 0, result);
24 }
25
26
27
28 console.log(`Generalized abbreviation are: ${generate_generalized_abbreviation('BAT')}`);

```

RUN
SAVE
RESET
Close

Output 2.148s

```
Generalized abbreviation are: 3,2T,1A1,1AT,B2,B1T,BA1,BAT
Generalized abbreviation are: 4,3e,2d1,2de,1o2,1o1e,1od1,1ode,c3,c2e,c1d1,cide,co2,cole,cod1,code
```

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

Back
Next

Balanced Parentheses (hard)
 Mark as Completed

Report an Issue
 Ask a Question

Create
Mark Course as Completed