Link to the problem

https://leetcode.com/problems/generalized-abbreviation/
Note: This problem is a premium LeetCode problem.

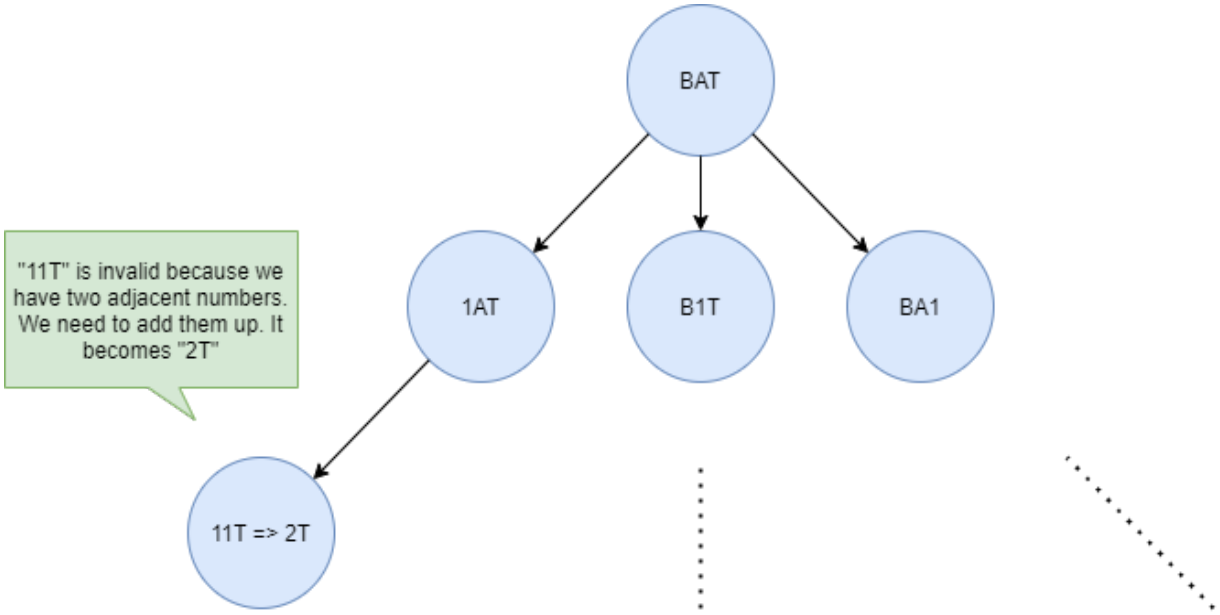
Requirements

What does the problem want? It wants us to “replace each substring of a given word with the count of characters in the substring. Return all possible abbreviations following this rule.”
A different way of interpreting this problem is: Suppose we have the following string of English letters

“BAT”

The problem wants us to try to replace all letters, one letter at a time, with the value “1”. However, if we replace one letter by “1” and we end up with two adjacent numbers, then we want to add them together.
Which means => Numbers get added together, letters do not.

For example, let’s take the word “BAT”:

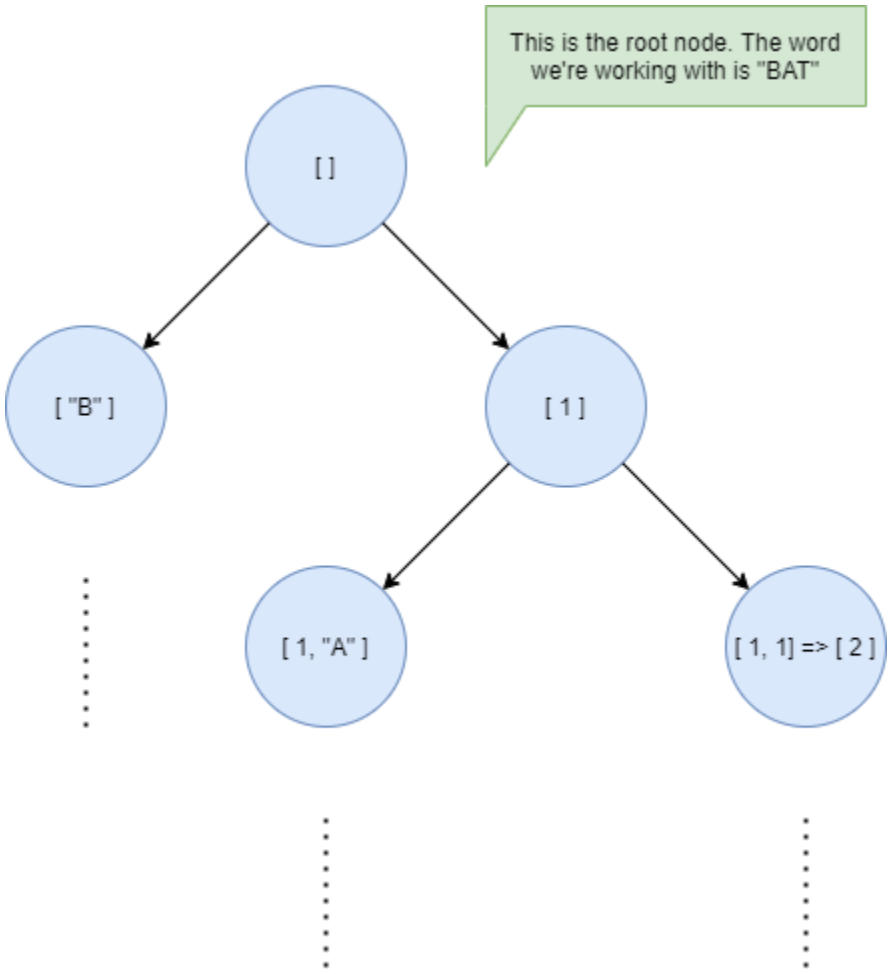


This is the same as “replacing each substring of the word with the count of characters in the substring.”.

Recursive solution

One possible way to solve this problem is to start with an empty list, then for every letter in the input word, we either add that letter to the list, or just add the number “1” instead of that letter. If two numbers are adjacent after we add a number, we sum them together.

For example, let’s take the word “BAT”



Once we reach a leaf node after going through all letters, we add the resulting list to the final array of resulting strings. This solution also requires backtracking where we revert what we did in one child node before visiting the next child node or before going back to the parent (either remove a letter from the array or minus 1 from the array).

Time and space complexity analysis

Time complexity

For every letter, we will have two scenarios: One where we keep it as a letter, and one where we replace it with the number ”1”. This means that at every level of the tree, we will have double the number of possibilities (this can be seen in the tree diagram above), and we will have a total of N levels, where N is the total number of letters in the input string.

- How many leaf nodes will we end up with? 2^N
- How much is the height/cost of every leaf node? N

The total cost is O(N * (2^N))

Space complexity

We have two sources of space complexity:

1. The call stack for the recursion
    a. As discussed in the time complexity, the height of the call stack tree is at most N
    b. The space complexity for this is N
2. The final array to store all the resulting strings
    a. How many final strings do we have? As many as we have leaf nodes in the tree
    b. How many leaf nodes do we have in the tree? 2^N nodes
    c. Ok, what is the size of every final string? They’re not all of the same size… However, the largest of them is of size N
    d. This gives us N * (2^N) space complexity, which is much bigger than the N from the call stack.

The final Space complexity is O(N * (2^N)) in a worst-case scenario.