320. Generalized Abbreviation

Summary

This article is for intermediate readers. It introduces the following ideas: Backtracking and Bit Manipulation

Solution

Approach #1 (Backtracking) [Accepted]

Intuition

How many abbreviations are there for a word of length n? The answer is 2^n because each character can either be abbreviated or not, resulting in different abbreviations.

Algorithm

The backtracking algorithm enumerates a set of partial candidates that, in principle, could be completed in several choices to give all the possible solutions to the problem. The completion is done incrementally, by extending the candidate in many steps. Abstractly, the partial candidates can be seen as nodes of a tree, the potential search tree. Each partial candidate is the parent of the candidates that derives from it by an extension step; the leaves of the tree are the partial candidates that cannot be extended any further.

In our problem, the partial candidates are incomplete abbreviations that can be extended by one of the two choices:

1. keep the next character;
2. abbreviate the next character.

We extend the potential candidate in a depth-first manner. We backtrack when we reach a leaf node in the search tree. All the leaves in the search tree are valid abbreviations and shall be put into a shared list which will be returned at the end.

Java Copy

Complexity Analysis

- Time complexity : $O(n2^n)$. For each call to `backtrack`, it either returns without branching, or it branches into two recursive calls. All these recursive calls form a complete binary recursion tree with 2^n leaves and $2^n - 1$ inner nodes. For each leaf node, it needs $O(n)$ time for converting builder to String; for each internal node, it needs only constant time. Thus, the total time complexity is dominated by the leaves. In total that is $O(n2^n)$.
- Space complexity : $O(n)$. If the return list doesn't count, we only need $O(n)$ auxiliary space to store the characters in `StringBuilder` and the $O(n)$ space used by system stack. In a recursive program, the space of system stack is linear to the maximum recursion depth which is n in our problem.

Approach #2 (Bit Manipulation) [Accepted]

Intuition

If we use 0 to represent a character that is not abbreviated and 1 to represent one that is. Then each abbreviation is mapped to an n bit binary number and vice versa.

Algorithm

To generate all the 2^n abbreviation with non-repetition and non-omission, we need to follow rules. In approach #1, the rules are coded in the backtracking process. Here we introduce another way.

From the intuition section, each abbreviation has a one to one relationship to a n bit binary number x. We can use these numbers as blueprints to build the corresponding abbreviations.

For example:

Given word = `"word"` and x = `0b0011`

Which means `'w'` and `'o'` are kept, `'r'` and `'d'` are abbreviated. Therefore, the result is "wo2".

Thus, for a number x, we just need to scan it bit by bit as if it is an array so that we know which character should be kept and which should be abbreviated.

To scan a number x bit by bit, one could extract its last bit by `b = x & 1` and shift x one bit to the right, i.e. `x >>= 1`. Doing this repeatedly, one will get all the n bits of x from last bit to first bit.

Java Copy

Complexity Analysis

- Time complexity : $O(n2^n)$. Building one abbreviation from the number x, we need scan all the n bits. Besides the `StringBuilder::toString` function is also linear. Thus, to generate all the 2^n, it costs $O(n2^n)$ time.
- Space complexity : $O(n)$. If the return list doesn't count, we only need $O(n)$ auxiliary space to store the characters in `StringBuilder`.

Comments: 7

Sort By

Type comment here... (Markdown is supported)

Preview Post

(/nyzhang)

NYZhang (/nyzhang) ★ 7 August 26, 2018 12:38 PM

This article is for intermediate readers [doge][doge][doge][doge]

0 Share Reply

(/sandadi)

Sandadi (/sandadi) ★ 4 May 8, 2018 3:32 PM

Can anyone help me in understanding why caching is bad for this problem. We can prune the recursion tree significantly if we cache the data and not allow the program traverse already traversed path.

0 Share Reply

SHOW 1 REPLY

(/cartesianist)

cartesianist (/cartesianist) ★ 16 February 1, 2018 11:21 PM

Regarding backtracking's general algorithm description, I think it would be better if the author either 1) referenced Wikipedia's backtracking article (a lot easier to follow) rather than just cutting and pasting some portion of it without citing it, or 2) wrote it in his own words, using his own thoughts.

2 Share Reply