

Dec. 9, 2016 | 11.8K views

Average Rating: 3.67 (12 votes)

Write a function to generate the generalized abbreviations of a word.

320. Generalized Abbreviation

Note: The order of the output does not matter.

Example:

```
Input: "word"
Output:
["word", "lord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "w02", "101d", "10r1", "w1r1",
```

Summary

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

Solution

Intuition

Approach #1 (Backtracking) [Accepted]

either be abbreviated or not, resulting in different abbreviations. Algorithm

How many abbreviations are there for a word of length n? The answer is 2^n because each character can

The backtracking algorithm enumerates a set of partial candidates that, in principle, could be completed in

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

several choices to give all the possible solutions to the problem. The completion is done incrementally, by

choices: 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.

```
Copy Copy
Java
 1 public class Solution {
       public List<String> generateAbbreviations(String word){
          List<String> ans = new ArrayList<String>();
          backtrack(ans, new StringBuilder(), word, 0, 0);
           return ans;
 6
 8
       // i is the current position
 9
       // k is the count of consecutive abbreviated characters
10
       private void backtrack(List<String> ans, StringBuilder builder, String word, int i, int k){
11
          int len = builder.length(); // keep the length of builder
         if(i == word.length()){
12
13
               if (k != 0) builder.append(k); // append the last k if non zero
14
               ans.add(builder.toString());
15
           } else {
          // the branch that word.charAt(i) is abbreviated
16
17
              backtrack(ans, builder, word, i + 1, k + 1);
18
19
             // the branch that word.charAt(i) is kept
20
              if (k != 0) builder.append(k);
21
              builder.append(word.charAt(i));
22
               backtrack(ans, builder, word, i + 1, 0);
23
24
           builder.setLength(len); // reset builder to the original state
25
        }
26 }
```

ullet Time complexity : $O(n2^n)$. For each call to ${f backtrack}$, it either returns without branching, or it

Complexity Analysis

- 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

should be kept and which should be abbreviated.

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.

can use these numbers as blueprints to build the corresponding abbreviations.

public List<String> generateAbbreviations(String word) {

For example:

From the intuition section, each abbreviation has a one to one relationship to a n bit binary number x. We

Java

1 public class Solution {

Given word = "word" and x = 0b0011

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

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. $\times >= 1$. Doing this repeatedly, one will get all the n bits of x from last bit to first bit. Copy Copy

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

```
List<String> ans = new ArrayList<>();
           for (int x = 0; x < (1 << word.length()); ++x) // loop through all possible x
                ans.add(abbr(word, x));
   6
            return ans;
   7
        }
   8
  9
       // build the abbreviation for word from number x
       private String abbr(String word, int x) {
  10
 11
           StringBuilder builder = new StringBuilder();
 12
           int k = 0, n = word.length(); // k is the count of consecutive ones in x
  13
            for (int i = 0; i < n; ++i, x >>= 1) {
                 if ((x & 1) == 0) { // bit is zero, we keep word.charAt(i)
  14
 15
                    if (k != 0) { // we have abbreviated k characters
                        builder.append(k);
 16
 17
                         k = 0; // reset the counter k
 18
                    }
 19
                    builder.append(word.charAt(i));
                 }
 20
  21
                 else // bit is one, increase k
  22
  23
 24
             if (k != 0) builder.append(k); //don't forget to append the last k if non zero
 25
             return builder.toString();
 26
         }
 27 }
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.

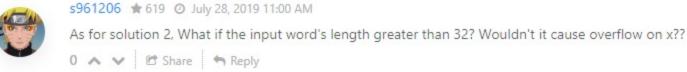
Comments: 10

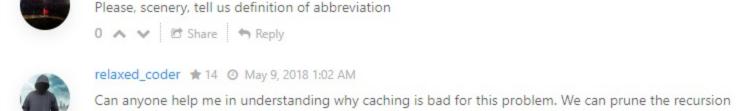
- Rate this article: * * * * *
- O Previous Next 0

Type comment here... (Markdown is supported)

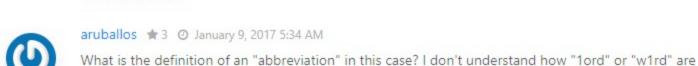
Sort By ▼

```
Preview
                                                                                                Post
sofs1 * 595 • May 5, 2019 5:23 PM
What is meant by generalized abbreviation? i can't understand the question completely. The example is
not self-explanatory.
8 A V C Share  Reply
SHOW 3 REPLIES
MitchellHe # 223 @ June 23, 2017 2:28 AM
I am afraid the 2nd solution is limited to the range of int. It will overflow when input is too long.
7 A V C Share  Reply
cartesianist # 45 @ February 2, 2018 9:51 AM
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.
6 A V C Share  Reply
LArch * 34 O November 8, 2017 7:57 AM
Very bad signature to use i and k.
1 A V C Share  Reply
```





tree significantly if we cache the data and not allow the program traverse already traversed path.



strings (strings, starting at startldx).

-2 ∧ ∨ 🗈 Share 🦘 Reply

-3 ∧ ∨ 🗈 Share 🦘 Reply

SHOW 2 REPLIES

http://www.dictionary.com/browse/abbreviation Ark-kun * 68 O November 30, 2017 4:51 PM

I think I've managed to do this in proper O(2^n) time and space. I just store the abbreviations of lesser

NYZhang * 35 • August 26, 2018 10:08 PM This article is for intermediate readers [doge][doge][doge][doge]

abbrevations of "word", using this definition of "abbrevation":