Generalized Abbreviation

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

Example:

Given word = `"word"`, return the following list (order does not matter):

```
["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1
", "1o2", "2r1", "3d", "w3", "4"]
```

Solution 1

The idea is: for every character, we can keep it or abbreviate it. To keep it, we add it to the current solution and carry on backtracking. To abbreviate it, we omit it in the current solution, but increment the count, which indicates how many characters have we abbreviated. When we reach the end or need to put a character in the current solution, and count is bigger than zero, we add the number into the solution.

```
  public List<String> generateAbbreviations(String word){
        List<String> ret = new ArrayList<String>();
        backtrack(ret, word, 0, "", 0);

        return ret;
    }

    private void backtrack(List<String> ret, String word, int pos, String cur, in
t count){
        if(pos==word.length()){
            if(count > 0) cur += count;
            ret.add(cur);
        }
        else{
            backtrack(ret, word, pos + 1, cur, count + 1);
            backtrack(ret, word, pos+1, cur + (count>0 ? count : "") + word.charA
t(pos), 0);
        }
    }
```

written by soymuybien original link here

Solution 2

```
public class Solution {
    public List<String> generateAbbreviations(String word) {
        List<String> res = new ArrayList<String>();
        int len = word.length();
        res.add(len==0 ? "" : String.valueOf(len));
        for(int i = 0 ; i < len ; i++)
            for(String right : generateAbbreviations(word.substring(i+1))){
                String leftNum = i > 0 ? String.valueOf(i) : "";
                res.add( leftNum + word.substring(i,i + 1) + right );
            }
        return res;
    }
}
```

written by caiqi8877 original link here

Solution 3

For each char `c[i]`, either abbreviate it or not.

1. Abbreviate: count accumulate `num` of abbreviating chars, but don't append it yet.
2. Not Abbreviate: append accumulated `num` as well as current char `c[i]`.
3. In the end append remaining `num`.
4. Using `StringBuilder` can decrease `36.4%` time.

This comes to the pattern I find powerful:

```
int len = sb.length(); // decision point
... backtracking logic ...
sb.setLength(len);     // reset to decision point
```

Similarly, check out remove parentheses and add operators.

```
public List<String> generateAbbreviations(String word) {
    List<String> res = new ArrayList<>();
    DFS(res, new StringBuilder(), word.toCharArray(), 0, 0);
    return res;
}

public void DFS(List<String> res, StringBuilder sb, char[] c, int i, int num) {
    int len = sb.length();
    if(i == c.length) {
        if(num != 0) sb.append(num);
        res.add(sb.toString());
    } else {
        DFS(res, sb, c, i + 1, num + 1);               // abbr c[i]

        if(num != 0) sb.append(num);                   // not abbr c[i]
        DFS(res, sb.append(c[i]), c, i + 1, 0);
    }
    sb.setLength(len);
}
```

written by yavinci original link here