CTCI 1.6: String Compression

October 25, 2017

Question

Implement a method to perform basic string compression using the counts of repeated characters. For example, the string aabcccccaaa would become a2blc5a3. If the "compressed"string would not become smaller than the original string, your method should return the original string. You can assume the string has only uppercase and lowercase letters (a - z).

Explanation and Algorithm

When first reading this question, it seems straight forward: iterate through a string, copy characters to a new string and count repeats. At each iteration, check if your current character the same as the next. If not, add this compressed version to the new string. There could be a more efficient method, though. What if instead we checked for length beforehand and used a StringBuilder class to define the compressed string? This way, we can create the compressed string first and return the shorter string: the input string or the compressed string. This is optimal if there isn't a large amount of repeating characters.

Hints

1. Those of you new to strings, how will you loop through a string? What properties can you use to get numbers (index of a particular character or .charAt will be useful)
2. Once you figure out the brute force solution, think of how you can use StringBuilder and make another method.
3. Use this method to your advantage to separate the compressed string from the input!

Code

```
/*Brute force*/

String compress(String str){
   String compressedString = "";
    int repeats = 0;
   for(int i=0; i<str.length; i++){
      repeats++;

    //if next character is different than current add this to char result
      if(i+1 >= str.length ||str.charAt(i) != str.charAt(i+1)){
         compressedString += str.charAt(i)+repeats;
            repeats = 0;
      }
    }
}

/*Optimal Method*/
String compress (String str) {

//check final length and return input string if it would be longer
int finalLength = countCompression(str);
if (finalLength )= str.length){
   return str;
}
StringBuilder compressed = new StringBuilder(finalLength); //initial
    capacity
int countConsecutive = 0;
for (int i = 0; i < str.length(); i++) {
   countConsecutive++;

   //if next character is different than current, append this char to
       result
   if (i + 1 >= str.lengthO || str.charAt(i) != str.charAt(i + 1) {
      compressed.append(str.charAt(i);
      compressed.append(countConsecutive);
      countConsecutive = 0;
   }
}
return compressed.toString();
}

int countCompression(String str) {
   int compressedLength = 0;
   int countConsecutive = 0;
   for (int i = 0; i < str.length(); i++) {
      countConsecutive++;
//if next character is different than current, increase the length
if ((i + 1 )= str.lengthO || str.charAt(i) != str.charAt(i + 1)) {
   compressedLength += 1 + String.valueOf(countConsecutive).length();
```

```
   countConsecutive = 0;
    }
}
return compressedLength;
}
```

Run time analysis

Time complexity for brute force is $O(p + k^2)$ where p is the size of the original string and m is the amount of character sequences. For the optimal solution, it's just $O(k^2)$.

Sources

.