Question 8.7 from CTCI: Permutations Without Duplicates

Benjamin De Brasi

February 28, 2018

Question

Write a method to compute all permutations of a string of unique characters.

Explanation and Algorithm

This problem should scream recursion. I will cover two ways to solve it recursively.

One way to solve it is to imagine you have all the permutations of previous the entire string up a certain letter. How can you add in 1 letter to all the permutations of the string? You need to add the current letter to every possible position in every previous permutation. So if you're trying to find all the permutations of "abc" and you have all the permutations of "ab" which are "ab" and "ba." What you want to do is insert 'c' into each position it could possibly be in which would be 'c' + "ab", "a" + 'c' + "b" and "ab" + 'c'.

The above approach involved taking the current character you are up to, placing into every position possible per recursive call. Another way to look at it is instead of placing each letter in each position is to hold constant the position of the current letter (as the first position for simplcity sake) and add that letter to all permutations thus far. Then move on to the next letter and repeat till all letters have been added.

Hints

1. Approach 1: Suppose you had all permutations of abc . How can you use that to get all permutations of abcd?
2. Approach 1 :The permutations of abc represent all ways of ordering abc. Now, we want to create all orderings of abed . Take a specific ordering of abed, such as bdea. This bdca string represents an ordering of abc, too:

Remove the d and you get bca. Given the string bca, can you create all the "related" orderings that include d, too?

3. Approach 1: Given a string such as bca, you can create all permutations of abed that have a J b J c in the order bca by inserting d into each possible location: dbca, bdca, bcda, bcad . Given all permutations of abc, can you then create all permutations of abcd?

4. Approach 1: You can create all permutations of abcd by computing all permutations of abc and then inserting d into each possible location within those.

5. Approach 2: If you had all permutations of t wo-character substrings, could you generate all permutations of three-character substrings?

6. Approach 2: To generate a permutation of abed, you need to pick an initial character. It can be a, b, c, or d. You can then permute the remaining characters. How can you use this approach to generate all permutations of the full string?

7. Approach 2: To generate all permutations of abcd, pick each character (a, b, c, or d) as a starting character. Permute the remaining characters and prepend the starting character. How do you permute the remaining characters? With a recursive process that follows the same logic.

8. Approach 2: You can implement this approach by having the recursive function pass back the list of the strings, and then you prepend the starting character to it. Or, you can push down a prefix to the recursive calls.

Code

```
/*Answer 1 */

ArrayList<String> getPerms(String remainder){
   if(str == null) return null;

   ArrayList<String> permutations = new ArrayList<String>();
   if(str.length() == 0){
      permutations.add("");
      return permutations;
   }

   char first = str.charAt(0);
   String remainder = str.substring(1);
   ArrayList<String> words = getPerms(remainder);
   for (String word : words){
      for(int j = 0; j <= word.length(); j++){
```

```
            String s = insertCharAt(word,first,j);
            permutations.add(s);
        }
    }
    return permutations;
}

String insertCharAt(String word, char c, int i){
    String start = word.substring(0,i);
    String end = word.substring(i);
    return start + c + end;
}

\begin{lstlisting}

/* Answer 2 */

ArrayList<String> getPerms(String remainder){
    int len = remainder.length();
    ArrayList<String> result = new ArrayList<String>();

    if(len == 0){
        result.add("");
        return result;
    }

    for(int i = 0; i < len; i++){
        String before = remainder.substring(0,i);
        String after = remainder.substring(i+1,len);
        ArrayList<String> partials = getPerms(before+after);

        for(String s : partials)
            result.add(remainder.charAt(i) + s);
    }

    return result;
}
```

Big O analysis

O(n!)

Sources

Question, answer and other material taken from Cracking the Coding Interview 6th edition by Gayle Laakmann McDowell.