String Permutations by changing case (medium)

We'll cover the following

Problem Statement

Given a string, find all of its permutations preserving the character sequence but changing case.

Example 1:

```
Input: "ad52"
Output: "ad52", "Ad52", "aD52", "AD52"
```

Example 2:

```
Input: "ab7c"
Output: "ab7c", "Ab7c", "aB7c", "AB7c", "ab7C", "Ab7C", "aB7C", "AB7C"
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```java
import java.util.*;

class LetterCaseStringPermutation {

  public static List<String> findLetterCaseStringPermutations(String str) {
    List<String> permutations = new ArrayList<>();
    // TODO: Write your code here
    return permutations;
  }

  public static void main(String[] args) {
    List<String> result = LetterCaseStringPermutation.findLetterCaseStringPermutations("ad52");
    System.out.println(" String permutations are: " + result);

    result = LetterCaseStringPermutation.findLetterCaseStringPermutations("ab7c");
    System.out.println(" String permutations are: " + result);
  }
}
```

Run | Save | Reset

Solution

This problem follows the Subsets pattern and can be mapped to Permutations.

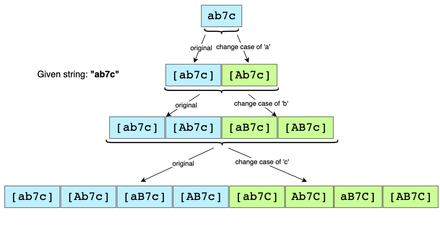
Let's take Example-2 mentioned above to generate all the permutations. Following a **BFS** approach, we will consider one character at a time. Since we need to preserve the character sequence, we can start with the actual string and process each character (i.e., make it upper-case or lower-case) one by one:

1. Starting with the actual string: `"ab7c"`
2. Processing the first character ('a'), we will get two permutations: `"ab7c", "Ab7c"`
3. Processing the second character ('b'), we will get four permutations: `"ab7c", "Ab7c", "aB7c", "AB7c"`
4. Since the third character is a digit, we can skip it.
5. Processing the fourth character ('c'), we will get a total of eight permutations: `"ab7c", "Ab7c", "aB7c", "AB7c", "ab7C", "Ab7C", "aB7C", "AB7C"`

Let's analyze the permutations in the 3rd and the 5th step. How can we generate the permutations in the 5th step from the permutations in the 3rd step?

If we look closely, we will realize that in the 5th step, when we processed the new character ('c'), we took all the permutations of the previous step (3rd) and changed the case of the letter ('c') in them to create four new permutations.

Here is the visual representation of this algorithm:



Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```java
import java.util.*;

class LetterCaseStringPermutation {

  public static List<String> findLetterCaseStringPermutations(String str) {
    List<String> permutations = new ArrayList<>();
    if (str == null)
      return permutations;

    permutations.add(str);
    // process every character of the string one by one
    for (int i = 0; i < str.length(); i++) {
      if (Character.isLetter(str.charAt(i))) { // only process characters, skip digits
        // we will take all existing permutations and change the letter case appropriately
        int n = permutations.size();
        for (int j = 0; j < n; j++) {
          char[] chs = permutations.get(j).toCharArray();
          // if the current character is in upper case change it to lower case or vice versa
          if (Character.isUpperCase(chs[i]))
            chs[i] = Character.toLowerCase(chs[i]);
          else
            chs[i] = Character.toUpperCase(chs[i]);
          permutations.add(String.valueOf(chs));
        }
      }
    }
    return permutations;
  }
```

Run | Save | Reset

Time complexity

Since we can have 2^N permutations at the most and while processing each permutation we convert it into a character array, the overall time complexity of the algorithm will be $O(N * 2^N)$.

Space complexity

All the additional space used by our algorithm is for the output list. Since we can have a total of $O(2^N)$ permutations, the space complexity of our algorithm is $O(N * 2^N)$.