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++

```
const find_letter_case_string_permutations = function(str) {
  permutations = [];
  // TODO: Write your code here
  return permutations;
};


console.log(`String permutations are: ${find_letter_case_string_permutations("ad52")}`)
console.log(`String permutations are: ${find_letter_case_string_permutations("ab7c")}`)

```

RUN SAVE RESET

Solution

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

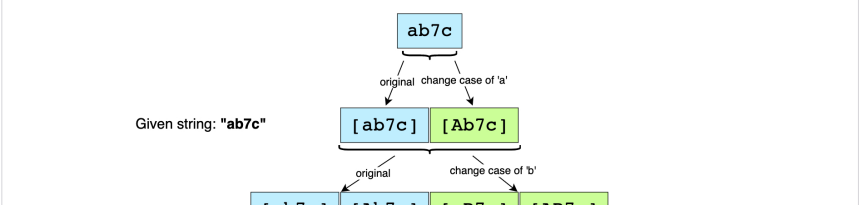
Let's take Example-2 mentioned above to generate all the permutations. Following a **BFS** approach, we will consider one character at a time. Since we need to preserve the character sequence, we can start with the actual string and process each character (i.e., make it upper-case or lower-case) one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

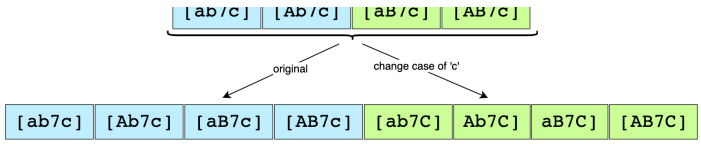
1. Starting with the actual string: `"ab7c"`
2. Processing the first character ('a'), we will get two permutations: `"ab7c", "Ab7c"`
3. Processing the second character ('b'), we will get four permutations: `"ab7c", "Ab7c", "aB7c", "AB7c"`
4. Since the third character is a digit, we can skip it.
5. Processing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

```
function find_letter_case_string_permutations(str) {
  permutations = [];
  permutations.push(str);
  // process every character of the string one by one
  for (i = 0; i < str.length; i++) {
    if (isNaN(parseInt(str[i], 10))) { // only process characters, skip digits
      // we will take all existing permutations and change the letter case appropriately
      const n = permutations.length;
      for (j = 0; j < n; j++) {
        const chs = permutations[j].split(''); // string to array
        // if the current character is in upper case, change it to lower case or vice versa
        if (chs[i] === chs[i].toLowerCase()) {
          chs[i] = chs[i].toUpperCase();
        } else {
          chs[i] = chs[i].toLowerCase();
        }
        permutations.push(chs.join(''));
      }
    }
  }

  return permutations;
}


console.log(`String permutations are: ${find_letter_case_string_permutations('ad52')}`);
console.log(`String permutations are: ${find_letter_case_string_permutations('ab7c')}`);
```

RUN SAVE RESET

Close

Output 2.351s

```
String permutations are: ad52,Ad52,aD52,AD52
String permutations are: ab7c,Ab7c,aB7c,AB7c,ab7C,Ab7C,aB7C,AB7C
```

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 $O(N * 2^N)$.