Letter Case Permutation

Given a string S, we can transform every letter individually to be lowercase or uppercase to create another string. Return a list of all possible strings we could create.

```
Examples:
Input: S = "a1b2"
Output: ["a1b2", "a1B2", "A1b2", "A1B2"]

Input: S = "3z4"
Output: ["3z4", "3Z4"]

Input: S = "12345"
Output: ["12345"]
```

Note:

- `S` will be a string with length at most `12`.
- `S` will consist only of letters or digits.

Solution 1

When I saw a problem, my first step is to draw a figure. See the figure below:
`abc`
`abc Abc` 0
`abc aBc Abc ABc` 1
`abc abC aBc aBC Abc AbC ABc ABC` 2

There we go! Is that a typical BFS/DFS problem? Yes, you are right!
Be careful to check whether a character is a digit or a letter(lower case or upper case).

```
class Solution {
    public List<String> letterCasePermutation(String S) {
        if (S == null) {
            return new LinkedList<>();
        }

        Queue<String> queue = new LinkedList<>();
        queue.offer(S);

        for (int i = 0; i < S.length(); i++) {
            if (Character.isDigit(S.charAt(i))) continue;
            int size = queue.size();
            for (int j = 0; j < size; j++) {
                String cur = queue.poll();
                char[] chs = cur.toCharArray();

                chs[i] = Character.toUpperCase(chs[i]);
                queue.offer(String.valueOf(chs));

                chs[i] = Character.toLowerCase(chs[i]);
                queue.offer(String.valueOf(chs));
            }
        }

        return new LinkedList<>(queue);
    }
}
```

written by flagbigoffer original link here

Solution 2

Let N be the number of letters in input, for each letter, we can toggle its case to get a new string. That is, there are 2 options for each letter: upper and lower cases. Therefore, we can generate 2 ^ N strings totally.

The details are as follows:

1. Add input into list.
2. Iterate through input string, when encountering a) a letter, toggle the case of the corresponding letter in all strings in the current list and append all new strings to list; b) a digit, ignore it.

```
    public List<String> letterCasePermutation(String S) {
        List<String> ans = new ArrayList<>(Arrays.asList(S));
        for (int i = 0; i < S.length(); ++i) { // Traverse string S.
            for (int j = 0, sz = ans.size(); S.charAt(i) > '9' && j < sz; ++j) { // 
S.charAt(i) > '9': letter, not digit.
                char[] ch = ans.get(j).toCharArray(); // transform to char[] the st
ring @ j of ans.
                ch[i] += ch[i] < 'a' ? 'a' - 'A' : 'A' - 'a'; // toggle case of cha
rAt(i).
                ans.add(String.valueOf(ch)); // append to the end of ans.
            }
        }
        return ans;
    }
```

written by rock original link here

Solution 3

Hello!

Here's a standard DFS/Backtracking solution done in Java. To make it more understandable for beginners, I made the code extremely modularized and made sure to comment out the functions of the methods. I hope you guys like my solution :)

The idea is to recurse through each individual character in the String and add the changeCase() version of it as well as retaining the original letter. You then loop through each character via recursion until the end condition is satisfied, as described in runCheck().

```
public static List<String> letterCasePermutation(String s) {
    int len = s.length();
    List<String> list = new ArrayList<>();
    list.add(s);
    // BackTrack solution
    backtrack(s,list, new String(),0);
    // Lists are passed by reference, so return list in this function
    return list;
}

public static void backtrack(String s, List<String> list, String curr, int index){
    if(runCheck(list,curr,s.length())){
        return;
    }
    // StringBuffer to manipulate immutable Strings
    StringBuffer str = new StringBuffer();
    str.append(curr);
    for(int i = index; i<s.length(); i++){
        int num = (int) s.charAt(i);
        boolean check = (64<num && num<91) || (96<num && num <123);
        // If the character is an upper/lower case letter
        if(check){
            // Add change-cased letter
            str.append(changeCase(num));
            backtrack(s,list,str.toString(),i+1);
            // Delete new letter and revert (so both permutations included)
            str.delete(str.length()-1,str.length());
            str.append(s.charAt(i));
            // If end conditions are satisfied, add to list
            if(runCheck(list,str.toString(),s.length())){
                return;
            }
        }
        // ELSE accounts for non-letter characters 0-9
        else {
       // Appends the number
            str.append(s.charAt(i));
            // If end conditions are satisfied, add to list
            if(runCheck(list,str.toString(),s.length())){
                return;
            }
        }
```

```
    }
}
// Passes in ASCII index number
// Returns upper case if num is lower case index, vice versa
public static char changeCase(int num){
    char c;
    // 'A'-'a' = 32
    if(64<num && num<91){
        num+=32;
    }
    else{
        num-=32;
    }
    c = (char) num;
    return c;
}

// Checks if list contains String s and if s is the correct length
public static boolean runCheck(List<String> list, String s, int n) {
    if(!list.contains(s) && s.length() == n) {
        list.add(s);
        return true;
    }
    return false;
}
```

written by matthewwang2020 original link here