Group Shifted Strings

Given a string, we can "shift" each of its letter to its successive letter, for example: `"abc" -> "bcd"`. We can keep "shifting" which forms the sequence:

```
"abc" -> "bcd" -> ... -> "xyz"
```

Given a list of strings which contains only lowercase alphabets, group all strings that belong to the same shifting sequence.

For example, given: `["abc", "bcd", "acef", "xyz", "az", "ba", "a", "z"]`, Return:

```
[
  ["abc","bcd","xyz"],
  ["az","ba"],
  ["acef"],
  ["a","z"]
]
```

Note: For the return value, each *inner* list's elements must follow the lexicographic order.

Solution 1

```
public class Solution {
    public List<List<String>> groupStrings(String[] strings) {
        List<List<String>> result = new ArrayList<List<String>>();
        Map<String, List<String>> map = new HashMap<String, List<String>>();
        for (String str : strings) {
            int offset = str.charAt(0) - 'a';
            String key = "";
            for (int i = 0; i < str.length(); i++) {
                char c = (char) (str.charAt(i) - offset);
                if (c < 'a') {
                    c += 26;
                }
                key += c;
            }
            if (!map.containsKey(key)) {
                List<String> list = new ArrayList<String>();
                map.put(key, list);
            }
            map.get(key).add(str);
        }
        for (String key : map.keySet()) {
            List<String> list = map.get(key);
            Collections.sort(list);
            result.add(list);
        }
        return result;
    }
}
```

written by stevenye original link here

Solution 2

The key to this problem is how to identify strings that are in the same shifting sequence. There are different ways to encode this.

In the following code, this manner is adopted: for a string `s` of length `n`, we encode its shifting feature as `"s[1] - s[0], s[2] - s[1], ..., s[n - 1] - s[n - 2],"`.

Then we build an `unordered_map`, using the above shifting feature string as key and strings that share the shifting feature as value. We store all the strings that share the same shifting feature in a `vector`. Well, remember to `sort` the `vector` since the problem requires them to be in lexicographic order :-)

A final note, since the problem statement has given that `"az"` and `"ba"` belong to the same shifting sequence. So if `s[i] - s[i - 1]` is negative, we need to add `26` to it to make it positive and give the correct result. BTW, taking the suggestion of @StefanPochmann, we change the difference from numbers to lower-case alphabets using `'a' + diff`.

The code is as follows.

```
class Solution {
public:
    vector<vector<string>> groupStrings(vector<string>& strings) {
        unordered_map<string, vector<string> > mp;
        for (string  s : strings)
            mp[shift(s)].push_back(s);
        vector<vector<string> > groups;
        for (auto m : mp) {
            vector<string> group = m.second;
            sort(group.begin(), group.end());
            groups.push_back(group);
        }
        return groups;
    }
private:
    string shift(string& s) {
        string t;
        int n = s.length();
        for (int i = 1; i < n; i++) {
            int diff = s[i] - s[i - 1];
            if (diff < 0) diff += 26;
            t += 'a' + diff + ',';
        }
        return t;
    }
};
```

written by jianchao.li.fighter original link here

Solution 3

Explanation

The basic idea is to set a key for each group: the sum of the difference between the adjacent chars in one string. Then we can easily group the strings belonging to the same shifting sequence with the same key. The code is as the following:

```
public List<List<String>> groupStrings(String[] strs) {
    HashMap<String, ArrayList<String>> map = new HashMap<String, ArrayList<String
>>();
    Arrays.sort(strs);
    for (String s : strs) {
        String key = "";
        for (int i = 1; i < s.length(); i++)
            key += String.format("%2d", (s.charAt(i) - s.charAt(i-1) + 26) % 26);
//Difference from the previous char.
        if (!map.containsKey(key)) map.put(key, new ArrayList<String>());
        map.get(key).add(s);
    }
    return new ArrayList<List<String>>(map.values());
}
```

written by Pixel_ original link here