522. Longest Uncommon Subsequence II

April 1, 2017 | 6.1K views
Average Rating: 4.29 (7 votes)

Given a list of strings, you need to find the longest uncommon subsequence among them. The longest uncommon subsequence is defined as the longest subsequence of one of these strings and this subsequence should not be **any** subsequence of the other strings.

A **subsequence** is a sequence that can be derived from one sequence by deleting some characters without changing the order of the remaining elements. Trivially, any string is a subsequence of itself and an empty string is a subsequence of any string.

The input will be a list of strings, and the output needs to be the length of the longest uncommon subsequence. If the longest uncommon subsequence doesn't exist, return -1.

Example 1:

```
Input: "aba", "cdc", "eae"
Output: 3
```

Note:

1. All the given strings' lengths will not exceed 10.
2. The length of the given list will be in the range of [2, 50].

Solution

Approach #1 Brute Force[Accepted]

In the brute force approach we will generate all the possible 2^n subsequences of all the strings and store their number of occurences in a hashmap. Longest subsequence whose frequency is equal to 1 will be the required subsequence. And, if it is not found we will return -1.

Java

```
public class Solution {
    public int findLUSLength(String[] strs) {
        HashMap < String, Integer > map = new HashMap < > ();
        for (String s: strs) {
            for (int i = 0; i < (1 << s.length()); i++) {
                String t = "";
                for (int j = 0; j < s.length(); j++) {
                    if ((i >> j) & 1) != 0)
                        t += s.charAt(j);
                }
                if (map.containsKey(t))
                    map.put(t, map.get(t) + 1);
                else
                    map.put(t, 1);
            }
        }
        int res = -1;
        for (String s: map.keySet()) {
            if (map.get(s) == 1)
                res = Math.max(res, s.length());
        }
        return res;
    }
}
```

Complexity Analysis

- Time complexity : $O(n * 2^x)$. where x is the average length of the strings and n is the total number of given strings.
- Space complexity : $O(n * 2^x)$. Hashmap of size $n * 2^x$ is used.

Approach #2 Checking Subsequence [Accepted]

Algorithm

By some analysis, we can note that if longest uncommon subsequence is there, then it will always be one of the string from the given list of strings. Using this idea, we can check each string that whether it is a subsequence of any other string. If a string is not a subsequence of any other string i.e. it is uncommon , we will return maximum length string out of them. If no string found, we will return -1.

To understand the method, look at the example given below:

res = -1

Java

```
public class Solution {
    public boolean isSubsequence(String x, String y) {
        int j = 0;
        for (int i = 0; i < y.length() && j < x.length(); i++)
            if (x.charAt(j) == y.charAt(i))
                j++;
        return j == x.length();
    }
    public int findLUSLength(String[] strs) {
        int res = -1;
        for (int i = 0, j; i < strs.length; i++) {
            for (j = 0; j < strs.length; j++) {
                if (j == i)
                    continue;
                if (isSubsequence(strs[i], strs[j]))
                    break;
            }
            if (j == strs.length)
                res = Math.max(res, strs[i].length());
        }
        return res;
    }
}
```

Complexity Analysis

- Time complexity : $O(x * n^2)$. where n is the number of strings and x is the average length of the strings.
- Space complexity : $O(1)$. Constant space required.

Approach #3 Sorting and Checking Subsequence [Accepted]

Algorithm

In the last approach, we needed to compare all the given strings and compare them for the subsequence criteria. We can save some computations if we sort the given set of strings based on their lengths initially.

In this approach, firstly we sort the given strings in decreasing order of their lengths. Then, we start off by comparing the longest string with all the other strings. If none of the other strings happens to be the subsequence of the longest string, we return the length of the longest string as the result without any need of further comparisons. If some string happens to be a subsequence of the longest string, we continue the same process by choosing the second largest string as the first string and repeat the process, and so on.

Java

```
public class Solution {
    public boolean isSubsequence(String x, String y) {
        int j = 0;
        for (int i = 0; i < y.length() && j < x.length(); i++)
            if (x.charAt(j) == y.charAt(i))
                j++;
        return j == x.length();
    }
    public int findLUSLength(String[] strs) {
        Arrays.sort(strs, new Comparator < String > () {
            public int compare(String s1, String s2) {
                return s2.length() - s1.length();
            }
        });
        for (int i = 0, j; i < strs.length; i++) {
            boolean flag = true;
            for (j = 0; j < strs.length; j++) {
                if (i == j)
                    continue;
                if (isSubsequence(strs[i], strs[j])) {
                    flag = false;
                    break;
                }
            }
            if (flag)
                return strs[i].length();
        }
        return -1;
    }
```

Complexity Analysis

- Time complexity : $O(x * n^2)$. where n is the number of strings and x is the average length of the strings.
- Space complexity : $O(1)$. Constant space required.

Rate this article:

Previous Next

Comments: 9 Sort By

Type comment here... (Markdown is supported)

Preview Post

donggua_fu ★ 187 April 3, 2017 6:11 PM
Maybe following sentence can be added in isSubsequence() function:

```
if(x.length()>y.length())
    return false;
```

4 Share Reply

sohammehta ★ 1157 January 8, 2018 7:02 PM
EDIT: After 2 hours i just realized that

Contiguous Subsequence is different than Subsequence that they are expecting here.
Read Question Properly. The above code will work for given case below.

Read More

3 Share Reply

SHOW 1 REPLY

mengqiyou ★ 83 December 26, 2017 10:05 PM
For method 3, the sort algorithm costs O(n) space. Arrays.sort() for Object use Merge sort which costs O(n).

0 Share Reply

SHOW 1 REPLY

DyXvLx5TAOadoD ★ 4655 July 10, 2017 11:53 AM
I don't quite understand this problem: if any string is a subsequence of itself, then a longest string couldn't be any subsequence of other strings because it is longer? Then it is the answer?
Can someone help me?
Thanks!

0 Share Reply

SHOW 2 REPLIES

base-2 ★ 0 April 10, 2017 9:55 AM
@vinod23. Thanks for your explanation .I was more curios about how you are deciding to sum up the individual characters to make the substring combination based on this check.

0 Share Reply

SHOW 1 REPLY

vinod23 ★ 461 April 10, 2017 2:28 AM
@priyeshnn-gmail.com Basically this condition checks the most significant bit(MSB) of i after shifting j bits right i.e. (i>>j). It will return true only if MSB of i >> j is equal to 1.

If n AND 1 is=0 then MSB of n is 0 else MSB is 1.

0 Share Reply

base-2 ★ 0 April 10, 2017 1:54 AM
@vinod23. Can you explain this logic a bit " if ((i >> j) & 1) != 0) "

0 Share Reply

SHOW 1 REPLY

vinod23 ★ 461 April 9, 2017 2:04 AM
@donggua_fu you are right. You can add for optimization.

0 Share Reply

s961206 ★ 753 May 14, 2020 5:39 AM
For method 1, shouldn't the complexity be n*(2^x)? You traverse the string s in the most inner loop??

0 Share Reply