161. One Edit Distance

Feb. 11, 2019 | 27.6K views

Average Rating: 4.75 (16 votes)

Previous Next

Given two strings ***s*** and ***t***, determine if they are both one edit distance apart.

Note:

There are 3 possiblities to satisify one edit distance apart:

1. Insert a character into ***s*** to get ***t***
2. Delete a character from ***s*** to get ***t***
3. Replace a character of ***s*** to get ***t***

Example 1:

```
Input: s = "ab", t = "acb"
Output: true
Explanation: We can insert 'c' into s to get t.
```

Example 2:

```
Input: s = "cab", t = "ad"
Output: false
Explanation: We cannot get t from s by only one step.
```

Example 3:

```
Input: s = "1203", t = "1213"
Output: true
Explanation: We can replace '0' with '1' to get t.
```

Solution

Approach 1: One pass algorithm

Intuition

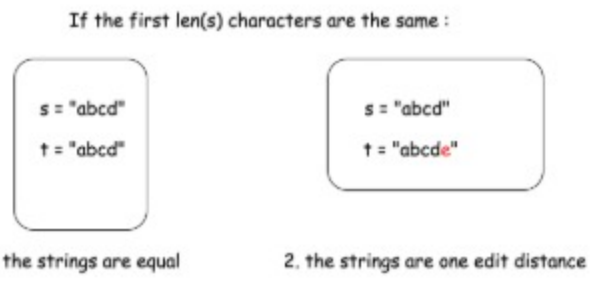
First of all, let's ensure that the string lengths are not too far from each other. If the length difference is 2 or more characters, then `s` and `t` couldn't be one edit away strings.

For the next let's assume that `s` is always shorter or the same length as `t`. If not, one could always call `isOneEditDistance(t, s)` to inverse the string order.

Now it's time to pass along the strings and to look for the first different character.

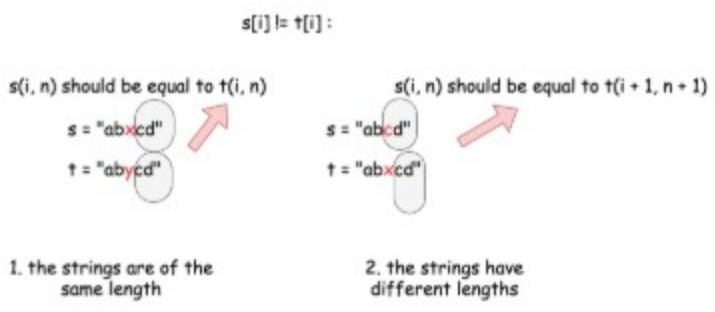
If there is no differences between the first `len(s)` characters, only two situations are possible :

- The strings are equal.
- The strings are one edit away distance.



Now what if there is a different character so that `s[i] != t[i]`.

- If the strings are of the same length, *all* next characters should be equal to keep one edit away distance. To verify it, one has to compare the substrings of `s` and `t` both starting from the `i + 1` th character.
- If `t` is one character longer than `s`, the additional character `t[i]` should be the only difference between both strings. To verify it, one has to compare a substring of `s` starting from the `i` th character and a substring of `t` starting from the `i + 1` th character.



Implementation

Java | Python — Copy

```java
class Solution {
  public boolean isOneEditDistance(String s, String t) {
    int ns = s.length();
    int nt = t.length();

    // Ensure that s is shorter than t.
    if (ns > nt)
      return isOneEditDistance(t, s);

    // The strings are NOT one edit away distance
    // if the length diff is more than 1.
    if (nt - ns > 1)
      return false;

    for (int i = 0; i < ns; i++)
      if (s.charAt(i) != t.charAt(i))
        // if strings have the same length
        if (ns == nt)
          return s.substring(i + 1).equals(t.substring(i + 1));
        // if strings have different lengths
        else
          return s.substring(i).equals(t.substring(i + 1));

    // If there is no diffs on ns distance
    // the strings are one edit away only if
    // t has one more character.
    return (ns + 1 == nt);
  }
}
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ in the worst case when string lengths are close enough `abs(ns - nt) <= 1`, where `N` is a number of characters in the longest string. $\mathcal{O}(1)$ in the best case when `abs(ns - nt) > 1`.
- Space complexity : $\mathcal{O}(N)$ because strings are immutable in Python and Java and to create substring costs $\mathcal{O}(N)$ space.

Problem generalization : Edit distance

Given two words word1 and word2, find the minimum number of operations required to convert word1 to word2.

Analysis written by @liaison and @andvary

Rate this article:

Previous Next

Comments: 13

Sort By

Type comment here... (Markdown is supported)

Preview Post

closewen ★ 23 June 30, 2019 11:19 PM

This solution is not O(1) memory, each s.substring will create a new string object.

13 Share Reply

SHOW 1 REPLY

milinthosani ★ 23 September 20, 2019 5:35 AM

My 2 pointer approach:
Time complexity: O(max(m,n))
Space complexity: O(1)

```
class Solution{
```

Read More

8 Share Reply

SHOW 1 REPLY

renjunyao ★ 21 August 12, 2019 12:08 PM Report

We can use two pointers, instead of substrings to make it O(1) space.

2 Share Reply

trlox48 ★ 1 March 25, 2020 7:40 AM

Logic is like this

- Get the length of both s & t strings as l1 and l2
- If they differ more than 1, they are more than one edit distance apart, so straight return false.
- Have two pointers i and j, starting with 0th index in s & t strings respectively and loop until they

Read More

1 Share Reply

Aj007 ★ 5 June 16, 2019 11:58 AM

Correct me if I am wrong but since a substring is created in the approach 1, this algo is O(N) space too since in Java, C# strings are immutable.

1 Share Reply

atulagrawal91 ★ 5 February 12, 2019 5:02 PM

```
if (Math.abs(s.length() - t.length()) > 1)
  return false;

int countChanges = 0;
```

Read More

1 Share Reply

ssl_91 ★ 25 August 5, 2019 8:14 AM

`return true` works fine for me. Curious to know when `(ns + 1 == nt)` will be executed?

In any, match or not match case, it should be already returned in the for loop, AFAIK.

0 Share Reply

SHOW 3 REPLIES

beauji_1116 ★ 0 April 14, 2020 10:19 AM

Why output should be false when input is "c" and "c"? It didn't say u cannot use the same character to replace the old one, so why i cannot use a letter "c" to replace itself?

0 Share Reply

Singhularity ★ 0 February 9, 2020 1:40 AM

Elegant solution, thank you. Why do you hate braces though, makes it so much harder to read and saves you virtually no time!

0 Share Reply

UNOboros ★ 0 October 1, 2019 11:15 PM

The solution is incorrect, consider the case s='bcd', t='abcd'. Then the first different position is 0 because you aligned a-b, b-c, c-d, and d-none. then you have s[0:] != t[1:].. But obviously it's not true...

0 Share Reply

SHOW 1 REPLY

‹ 1 2 ›