



Single Mutation

We'll determine how to tell if two different strings are similar or extremely different. We'll also cover the idea of the double-condition loop, another tool to add to our repertoire.

We'll cover the following



- Single Mutation
 - Instructions
 - Examples:
- Solution
 - How It Works
 - Checking Length
 - Looping Through
 - Options
 - Time
 - $O(n)$,
 - Space
 - $O(1)$
- Conclusion

Single Mutation#

Instructions#

There are 3 types of possible string mutations: character insertion, character deletion, or character substitution. Write a function that accepts 2 strings and returns `true` if the strings are the same except for 0 or 1 mutations. Return `false` otherwise.

Inputs: String, String

Output: Boolean

Examples:#

- Single Deletion:
 - 'abcd', 'abc'
 - 'abcd', 'acd'
- Single Insertion:
 - 'abcd', 'abcde'
 - 'abcd', 'abXcd'
- Single Substitution:
 - 'abcd', 'abXd'
 - 'abcd', 'Xbcd'

```
1 function singleMutation(str1, str2) {  
2     // Your code here  
3 }
```



Solution#



```
1 function singleMutation(str1, str2) {
2   if(Math.abs(str1.length - str2.length) > 1) {
3     return false;
4   }
5
6   let mutations = 0;
7
8   for(let i = 0, j = 0; i < str1.length || j < str2.length; i++, j++) {
9     if(str1[i] !== str2[j]) {
10      mutations++;
11      if(mutations > 1) {
12        return false;
13      }
14
15      if(str1.length > str2.length) {
16        j--;
17      } else if(str1.length < str2.length){
18        i--;
19      }
20    }
21  }
22
23  return true;
24 }
```

**Show Results****Show Console**

1.29s

**9 of 9 Public Tests Passed**

Result	Input	Expected Output	Actual Output	Reason
✓	singleMutation('abcd', 'abc')	true	true	Succeeded
✓	singleMutation('abcd', 'abXcd')	true	true	Succeeded

Result	Input	Expected Output	Actual Output 	Reason 
✓	<code>singleMutation('abcd', 'abXd')</code>	true	true	Succeeded
✓	<code>singleMutation('abcd', 'abXX')</code>	false	false	Succeeded
✓	<code>singleMutation('abcd', 'ab')</code>	false	false	Succeeded
✓	<code>singleMutation('abcd', 'abcdef')</code>	false	false	Succeeded
✓	<code>singleMutation('abcd', 'aXcX')</code>	false	false	Succeeded
✓	<code>singleMutation('abcd', 'aXc')</code>	false	false	Succeeded
✓	<code>singleMutation('abcd', 'aXcde')</code>	false	false	Succeeded

How It Works#

Checking Length#

In the first if-statement, we check if the string lengths are different by two or greater. If so, we know there are at least two deletions or additions and we can return `false`.

Looping Through#

The for-loop is more complex than those we've seen so far. We're maintaining indexes for both strings (`i` and `j`), but we're doing it in a single loop.

We're making sure that both indexes reach the end of their strings, and we're incrementing both indexes as well.

At each iteration, we check whether the characters in the strings *at their current index* are equivalent. If so, we do nothing and move on.



If not, we increase our `mutations` counter, as there is a mutation present. If this counter ever gets to be greater than `1`, we return `false`.

Options#

If the characters are not equivalent, there are three things we can do to proceed:



(/learn)

1. If the strings are the same length, we only increment our `mutations` counter and move on

2. If `str1` is longer than `str2`, we know that `str1` has an extra character and we've reached that character. We decrease `j` by 1.

Theoretically, the strings are now "lined up" until the end. If there are no more mutations, the function will proceed correctly.

3. If `str2` is longer than `str1`, the same logic applies, but we decrement `i` instead of `j`.

Time#

We'll only consider cases where the strings are either the same length or off by one. In this case, n will be the length of the strings.

If the string lengths are off by one, the time complexity is:

$O(n)$,

as we process the strings once. There's only one for-loop.

Space#

We don't store any data in this function.



$O(1)$ #

Conclusion#

We add yet another tool to our repertoire: the double-condition for-loop. These come in handy when we need to process different pieces of data simultaneously. If we find that we need to compare items at each iteration and we can't do it by looping through just one of the data sets, we might be able to apply a second condition to our loop.

← Back

Maximum Profits

Next →

All Anagrams



Mark as Completed



Report an Issue