# 243. Shortest Word Distance

April 11, 2016 | 33.6K views

Average Rating: 4.81 (41 votes)

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

#### Example: Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

```
Input: word1 = "coding", word2 = "practice"
Output: 3
Input: word1 = "makes", word2 = "coding"
Output: 1
```

### You may assume that word1 does not equal to word2, and word1 and word2 are both in the list.

Solution

Note:

## computed so far.

find all occurrences  $i_1$  and  $i_2$  of the two words, and check if  $|i_1-i_2|$  is less than the minimum distance Approach #1 (Brute Force) [Accepted]

This is a straight-forward coding problem. The distance between any two positions  $i_1$  and  $i_2$  in an array is

 $|i_1-i_2|$ . To find the shortest distance between word1 and word2, we need to traverse the input array and

### Algorithm

#### A naive solution to this problem is to go through the entire array looking for the first word. Every time we

find an occurrence of the first word, we search the entire array for the closest occurrence of the second word. Java

### public int shortestDistance(String[] words, String word1, String word2) {

```
int minDistance = words.length;
      for (int i = 0; i < words.length; i++) {</pre>
          if (words[i].equals(word1)) {
              for (int j = 0; j < words.length; j++) {</pre>
                  if (words[j].equals(word2)) {
                       minDistance = Math.min(minDistance, Math.abs(i - j));
              }
          }
      }
      return minDistance;
 }
Complexity Analysis
```

#### The time complexity is $O(n^2)$ , since for every occurrence of word1 , we traverse the entire array in search for the closest occurrence of word2.

Space complexity is O(1), since no additional space is used.

Approach #2 (One-pass) [Accepted]

# Algorithm

int i1 = -1, i2 = -1;

int minDistance = words.length;

#### We can greatly improve on the brute-force approach by keeping two indices i1 and i2 where we store the most recent locations of word1 and word2. Each time we find a new occurrence of one of the words, we do

not need to search the entire array for the other word, since we already have the index of its most recent occurrence. Java

public int shortestDistance(String[] words, String word1, String word2) {

```
int currentDistance;
      for (int i = 0; i < words.length; i++) {</pre>
          if (words[i].equals(word1)) {
              i1 = i;
          } else if (words[i].equals(word2)) {
               i2 = i;
          }
          if (i1 != -1 && i2 != -1) {
               minDistance = Math.min(minDistance, Math.abs(i1 - i2));
          }
      }
      return minDistance;
 }
Complexity Analysis
The time complexity is O(n). This problem is inherently linear; we cannot do better than O(n) because at
```

# Space complexity is O(1), since no additional space is allocated.

O Previous

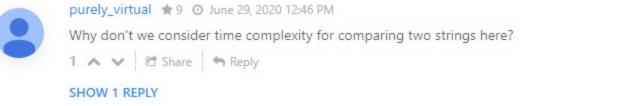
Analysis written by: @noran

the very least, we have to read the entire input.

Rate this article: \* \* \* \* \*

Next

```
Comments: 14
                                                                                                  Sort By ▼
              Type comment here... (Markdown is supported)
              Preview
                                                                                                    Post
             You do not need currentDistance here.
             56 ∧ ∨ ☑ Share ¬ Reply
             SHOW 1 REPLY
             Suralin ★ 161 ② September 10, 2018 10:40 PM
             Don't need either currentDistance or to keep track of both indices:
              min_dist = len(words)
              curr_word, idx = None, 0
                                                     Read More
             10 ∧ ∨ ☑ Share ← Reply
             SHOW 1 REPLY
             weijinwang ★ 5 ② September 21, 2017 11:17 PM
             Check words[i] is word1 or word2, else continue. This can save a little.
             5 A V @ Share  Reply
             Mr-Programmer ★ 30 ② June 2, 2019 5:52 AM
             Once we find the minDistance to be equal to 1, we can just return because it can never go down below
             that. This will help in saving some time as well if the first two words itself are word1 and word2.
             3 A V C Share Share
             SherMM ★ 59 ② February 7, 2018 1:41 AM
              def shortestDistance(self, words, word1, word2):
                  :type words: List[str]
                  :type word1: str
                                                     Read More
             2 A V 🗗 Share 👆 Reply
             purely_virtual ★ 9 ② June 29, 2020 12:46 PM
             Why don't we consider time complexity for comparing two strings here?
             1 A V 🗗 Share 🦘 Reply
```

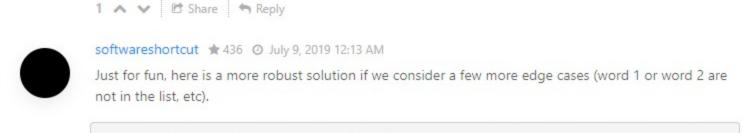


# out = first = second = float('inf')

My python submission

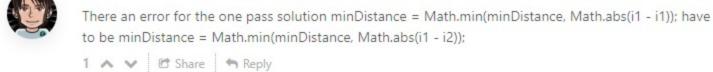
Mrmagician 🖈 332 🗿 April 2, 2020 2:22 AM

for i, word in enumerate (words): if word == word1: first =

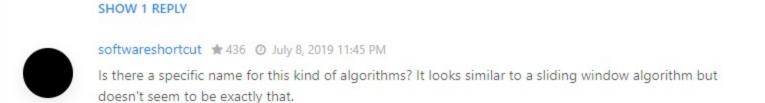


Read More

```
public int shortestDistance(String[] words, String word1, String word2) {
                                   Read More
1 A V Share Seply
```



jaumard # 1 @ April 17, 2016 1:14 AM



SHOW 1 REPLY

0 ∧ ∨ ☑ Share ¬ Reply