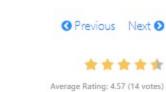
# 599. Minimum Index Sum of Two Lists \*\*

May 27, 2017 | 39.7K views



**6** 0 0

Suppose Andy and Doris want to choose a restaurant for dinner, and they both have a list of favorite restaurants represented by strings.

You need to help them find out their common interest with the least list index sum. If there is a choice tie between answers, output all of them with no order requirement. You could assume there always exists an answer.

```
Example 1:
 Input:
  ["Shogun", "Tapioca Express", "Burger King", "KFC"]
 ["Piatti", "The Grill at Torrey Pines", "Hungry Hunter Steakhouse", "Shogun"]
 Output: ["Shogun"]
 Explanation: The only restaurant they both like is "Shogun".
```

```
Example 2:
 Input:
 ["Shogun", "Tapioca Express", "Burger King", "KFC"]
 ["KFC", "Shogun", "Burger King"]
 Output: ["Shogun"]
```

### Note:

Explanation: The restaurant they both like and have the least index sum is "Shogun" wi

- The length of both lists will be in the range of [1, 1000]. 2. The length of strings in both lists will be in the range of [1, 30]. 3. The index is starting from 0 to the list length minus 1.
- No duplicates in both lists.

# Solution

# In this approach, we compare every string in list1 and list2 by traversing over the whole list list2 for every

Approach #1 Using HashMap [Accepted]

string chosen from list1. We make use of a hashmap map, which contains elements of the form (sum: $list_{sum}$  ). Here, sum refers to the sum of indices of matching elements and  $list_{sum}$  refers to the list of matching strings whose indices' sum equals sum. Thus, while doing the comparisons, whenever a match between a string at  $i^{th}$  index of list1 and  $j^{th}$  index of list2 is found, we make an entry in the map corresponding to the sum i+j, if this entry isn't already

present. If an entry with this sum already exists, we need to keep a track of all the strings which lead to the same index sum. Thus, we append the current string to the list of strings corresponding to sum i+j. At the end, we traverse over the keys of the map and find out the list of strings corresponding to the key reprsenting the minimum sum.

**Сору** Java

```
1 public class Solution {
         public String[] findRestaurant(String[] list1, String[] list2) {
             HashMap < Integer, List < String >> map = new HashMap < > ();
             for (int i = 0; i < list1.length; i++) {
                for (int j = 0; j < list2.length; j++) {
                    if (list1[i].equals(list2[j])) {
  7
                         if (|map.containsKey(i + j))
                            map.put(i + j, new ArrayList < String > ());
                         map.get(i + j).add(list1[i]);
  10
 11
                 }
  12
             int min_index_sum = Integer.MAX_VALUE;
 13
             for (int key: map.keySet())
  14
  15
                 min_index_sum = Math.min(min_index_sum, key);
  16
             String[] res = new String[map.get(min_index_sum).size()];
             return map.get(min_index_sum).toArray(res);
 17
  18
 19 }
  20
Complexity Analysis
```

# • Time complexity : $O(l_1 * l_2 * x)$ . Every item of list1 is compared with all the items of list2. $l_1$ and $l_2$

- are the lengths of list1 and list2 respectively. And x refers to average string length. Space complexity: O(l<sub>1</sub> \* l<sub>2</sub> \* x). In worst case all items of list1 and list2 are same. In that case,
- hashmap size grows upto  $l_1 * l_2 * x$ , where x refers to average string length.

## Algorithm

Approach #2 Without Using HashMap [Accepted]

### Another method could be to traverse over the various sum (index sum) values and determine if any such

string exists in list1 and list2 such that the sum of its indices in the two lists equals sum. Now, we know that the value of index sum, sum could range from 0 to m+n-1. Here, m and n refer to

the length of lists list1 and list2 respectively. Thus, we choose every value of sum in ascending order. For every sum chosen, we iterate over list1. Suppose, currently the string at  $i^{th}$  index in list1 is being considered. Now, in order for the index sum sum to be the one corresponding to matching strings in list1and list2, the string at index j in list2 should match the string at index i in list1, such that sum = i + j. Or, stating in other terms, the string at index j in list2 should be equal to the string at index i in list1, such that j=sum-i. Thus, for a particular sum and i(from list1), we can directly determine that we need to

Doing such checks/comparisons, iterate over all the indices of list1 for every sum value chosen. Whenver a match occurs between list1 and list2, we put the matching string in a list res. We do the same process of checking the strings for all the values of sum in ascending order. After

check the element at index j = sum - i in list2, instead of traversing over the whole list2.

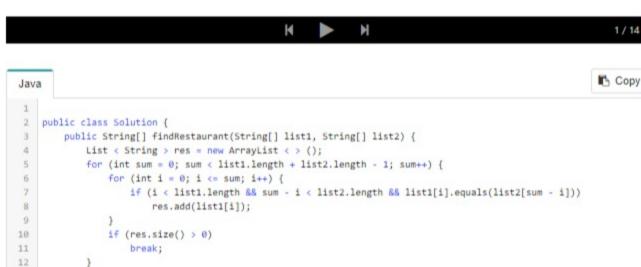
completing every iteration over list1 for a particular sum, we check if the res list is empty or not. If it is empty, we need to continue the process with the next sum value considered. If not, the current res gives the required list with minimum index sum. This is because we are already considering the index sum values in ascending order. So, the first list to be found is the required resultant list.

The following example depicts the process:

Shogun **Tapioca Express Burger King** list1

Shogun

Tapioca Express



## takes x time. Here, x refers to the average string length. Space complexity: O(r \* x). res list is used to store the result. Assuming r is the length of res.

**Complexity Analysis** 

13

14 15 } list2

KFC

return res.toArray(new String[res.size()]);

a mapping from the elements of list1 to their indices.

Approach #3 Using HashMap (linear) [Accepted]

• Time complexity :  $O((l_1+l_2)^2*x)$ . There are two nested loops upto  $l_1+l_2$  and string comparison

We make use of a HashMap to solve the given problem in a different way in this approach. Firstly, we traverse over the whole list1 and create an entry for each element of list1 in a HashMap map, of the form (list[i], i). Here, i refers to the index of the  $i^{th}$  element, and list[i] is the  $i^{th}$  element itself. Thus, we create

Now, we traverse over list2. For every element list2[j], of list2 encountered, we check if the same element already exists as a key in the map. If so, it means that the element exists in both list1 and list2. Thus, we find out the sum of indices corresponding to this element in the two lists, given by sum=map.get(list[j]) + j. If this sum is lesser than the minimum sum obtained till now, we update the

If the sum is equal to the minimum sum obtained till now, we put an extra entry corresponding to the

resultant list to be returned, res, with the element list2[j] as the only entry in it.

public String[] findRestaurant(String[] list1, String[] list2) {

HashMap < String, Integer > map = new HashMap < String, Integer > ();

Below code is inspired by @cloud.runner **Сору** Java

for (int i = 0; i < list1.length; i++) map.put(list1[i], i); List < String > res = new ArrayList < > (); int min\_sum = Integer.MAX\_VALUE, sum;

element list2[j] in the res list.

1 public class Solution {

```
if (map.containsKey(list2[j])) {
  10
                    sum = j + map.get(list2[j]);
                    if (sum < min_sum) {
  11
 12
                        res.clear();
 13
                        res.add(list2[j]);
  14
                        min_sum = sum;
  15
                    } else if (sum == min_sum)
  16
                        res.add(list2[j]);
 17
 18
 19
             return res.toArray(new String[res.size()]);
 20
 21 }
Complexity Analysis
  • Time complexity : O(l_1 + l_2). Every item of list2 is checked in a map of list1. l_1 and l_2 are the
     lengths of list1 and list2 respectively.
   • Space complexity : O(l_1 * x). hashmap size grows upto l_1 * x, where x refers to average string
     length.
Rate this article: * * * * *
```

# O Previous

Comments: (31)

Next **⊙** 

Sort By -

Post

A Report

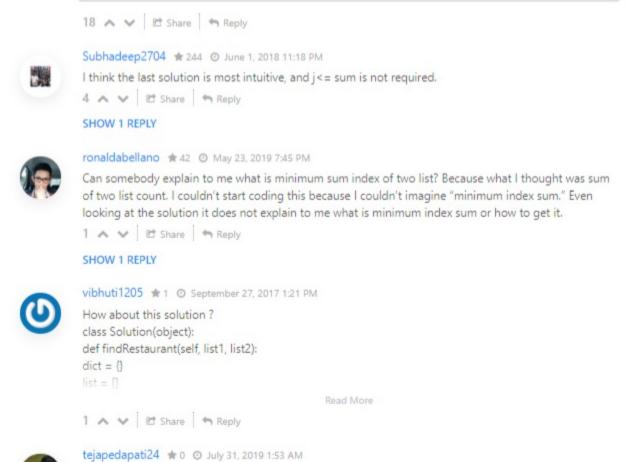
A Report

Type comment here... (Markdown is supported)

if (list1.length > list2.length) return findRestaurant(list2, list1);

ryanleitaiwan # 30 @ August 3, 2018 11:44 PM To reduce space complexity to O(min(I1, I2) \* x), we can force list1 to be no longer than list2.

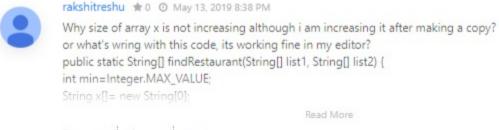
@ Preview





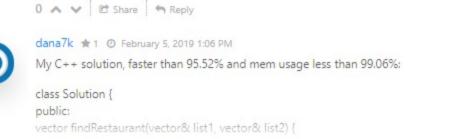
class Solution: def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]: dic={} Read More

Read More



0 A V & Share Share

0 ∧ ∨ Ø Share ♠ Reply







HashMap Solution

0 A V & Share Share

azimbabu \* 137 O October 12, 2018 2:18 PM

class Solution { public String[] findRestaurant(String[] list1, String[] list2) { Map<String. Integer> hashMap = new HashMap<>(): Read More

Why is time complexity of approach #3 is O(I1 + I2). Shouldn't it be O(max(I1, I2))?

(1234)