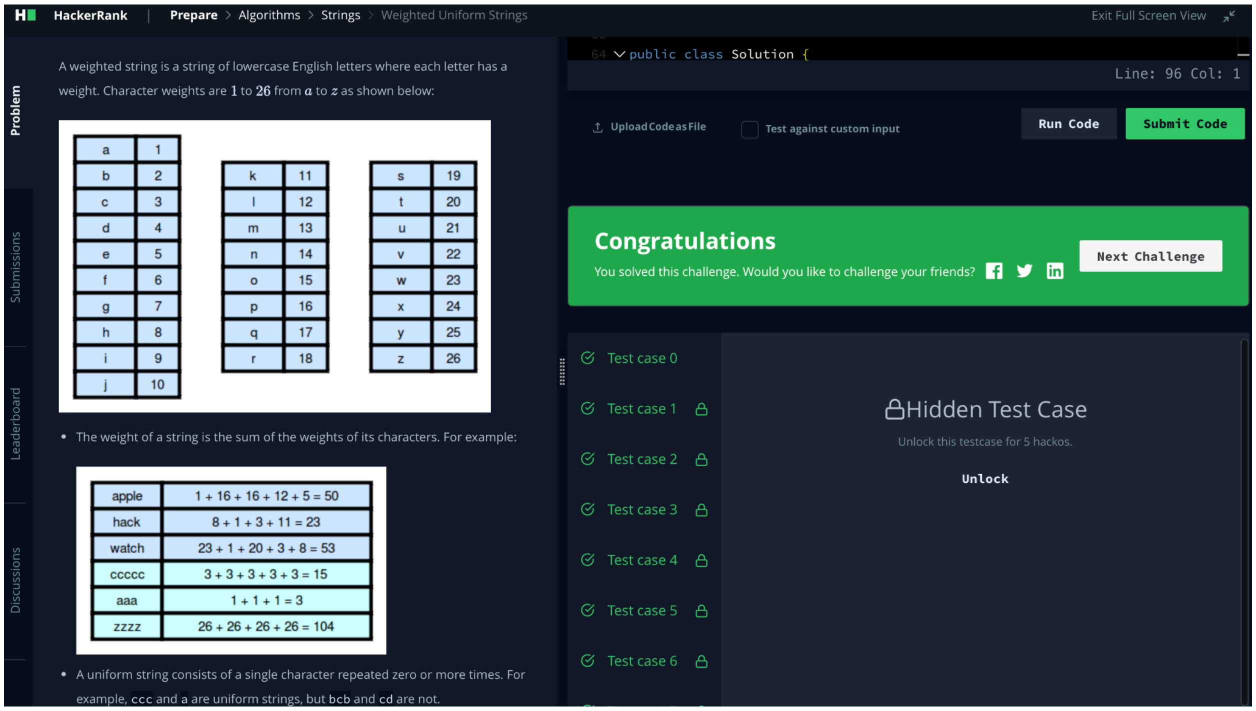
Weighted Uniform String



A weighted string is a string of lowercase English letters where each letter has a weight. Character weights are 1 to 26 from a to z as shown below:

a	1	k	11	s	19
b	2	l	12	t	20
c	3	m	13	u	21
d	4	n	14	v	22
e	5	o	15	w	23
f	6	p	16	x	24
g	7	q	17	y	25
h	8	r	18	z	26
i	9				
j	10				

- The weight of a string is the sum of the weights of its characters. For example:

apple	1 + 16 + 16 + 12 + 5 = 50
hack	8 + 1 + 3 + 11 = 23
watch	23 + 1 + 20 + 3 + 8 = 53
ccccc	3 + 3 + 3 + 3 + 3 = 15
aaa	1 + 1 + 1 = 3
zzzz	26 + 26 + 26 + 26 = 104

- A uniform string consists of a single character repeated zero or more times. For example, ccc and a are uniform strings, but bcb and cd are not.

```
public class Solution {
```

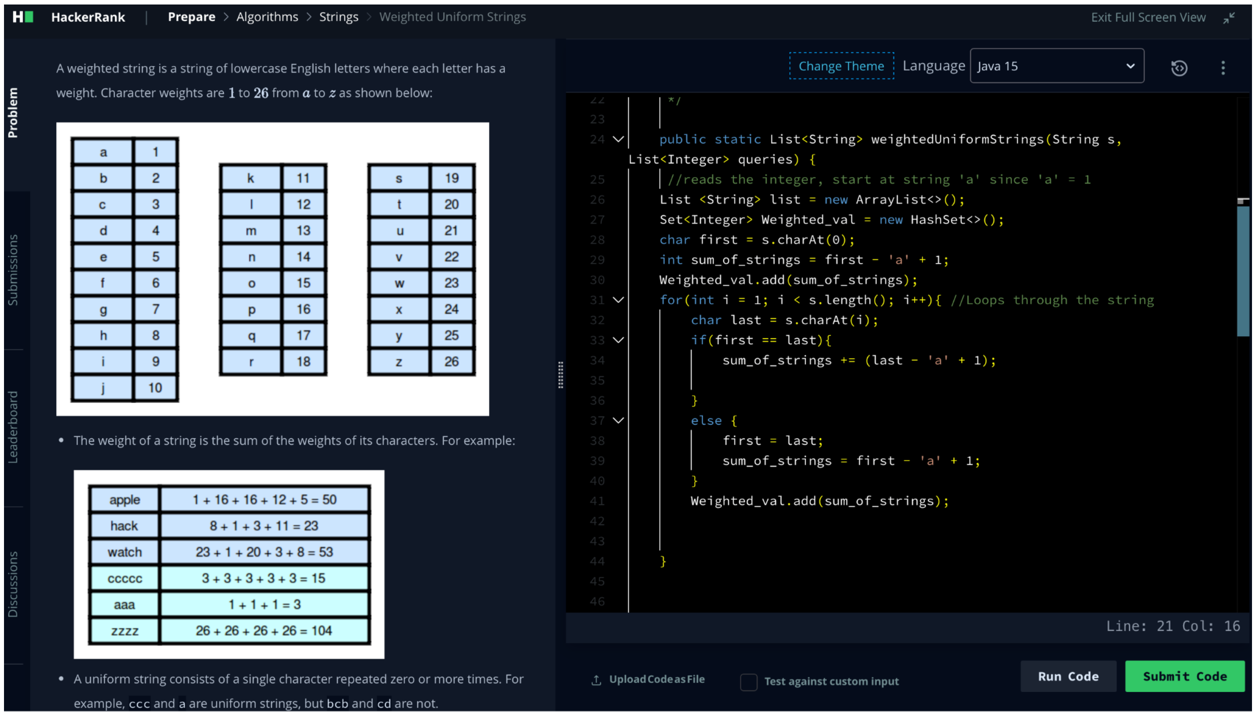
Line: 96 Col: 1

Congratulations

You solved this challenge. Would you like to challenge your friends?

Hidden Test Case

Unlock this testcase for 5 hackos.



A weighted string is a string of lowercase English letters where each letter has a weight. Character weights are 1 to 26 from a to z as shown below:

a	1	k	11	s	19
b	2	l	12	t	20
c	3	m	13	u	21
d	4	n	14	v	22
e	5	o	15	w	23
f	6	p	16	x	24
g	7	q	17	y	25
h	8	r	18	z	26
i	9				
j	10				

- The weight of a string is the sum of the weights of its characters. For example:

apple	1 + 16 + 16 + 12 + 5 = 50
hack	8 + 1 + 3 + 11 = 23
watch	23 + 1 + 20 + 3 + 8 = 53
ccccc	3 + 3 + 3 + 3 + 3 = 15
aaa	1 + 1 + 1 = 3
zzzz	26 + 26 + 26 + 26 = 104

- A uniform string consists of a single character repeated zero or more times. For example, ccc and a are uniform strings, but bcb and cd are not.

Language: Java 15

```
    */

    public static List<String> weightedUniformStrings(String s,
List<Integer> queries) {
    //reads the integer, start at string 'a' since 'a' = 1
    List <String> list = new ArrayList<>();
    Set<Integer> Weighted_val = new HashSet<>();
    char first = s.charAt(0);
    int sum_of_strings = first - 'a' + 1;
    Weighted_val.add(sum_of_strings);
    for(int i = 1; i < s.length(); i++){ //Loops through the string
        char last = s.charAt(i);
        if(first == last){
            sum_of_strings += (last - 'a' + 1);

        }
        else {
            first = last;
            sum_of_strings = first - 'a' + 1;
        }
        Weighted_val.add(sum_of_strings);


    }


```

Line: 21 Col: 16

The time and space complexity:
The time complexity for this code is separated into two categories, first part goes to the Array list time complexity, as the first for loop iterates through the string, and this time complexity can be denoted as O(n). The second for loop iterates through the queries list (for int j: queries) as this surfs through another list, the time complexity for this loop is O(x) as it is another separate list that it goes through. So the time complexity for this code is O(n+x).
Space complexity, the space complexity for this code is denoted by the Hash set, and another one goes to the array list, because in the code you deal with two list at the same time, and their space is denoted by “n”, (length of s), and then for the array list used it is dependent on the number of queries, I denoted by ‘x’. So as the time complexity the space complexity is the sum of both these space complexities. O(n + x).

*footnote: I used x as a variable to make sure that the time complexities depended on different things.

Time and Space complexity:
The time complexity is determined by the sorting method used to sort the list, and the time complexity for a quick sort like this is)(n log n), the for loops have a constraint, with it being no more than 5, since it only goes through a constant amounts of iteration the time complexity for this is O(1). Same can be said for the other for loop since it goes up to arr.size - 1 always, it has a constant time complexity as well. The overall time complexity of this code is denoted by the quick sort method used to sort the arrays as it dominates over the other two time complexities. The Time complexity is O(n log n)
The space complexity is determined by the size of the input, and since the size of the input is not affected by the variables the code has, its space complexity remains constant. The space complexity for this code is O(1).