

Fruits into Baskets (medium)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time Complexity
 - Space Complexity
- Similar Problems

Problem Statement

Given an array of characters where each character represents a fruit tree, you are given **two baskets**, and your goal is to put **maximum number of fruits in each basket**. The only restriction is that **each basket can have only one type of fruit**.

You can start with any tree, but you can't skip a tree once you have started. You will pick one fruit from each tree until you cannot, i.e., you will stop when you have to pick from a third fruit type.

Write a function to return the maximum number of fruits in both the baskets.

Example 1:


```
Input: Fruit=['A', 'B', 'C', 'A', 'C']
Output: 3
Explanation: We can put 2 'C' in one basket and one 'A' in the other from the subarray ['C', 'A', 'C']
```


Example 2:


```
Input: Fruit=['A', 'B', 'C', 'B', 'B', 'C']
Output: 5
Explanation: We can put 3 'B' in one basket and two 'C' in the other basket.
This can be done if we start with the second letter: ['B', 'C', 'B', 'B', 'C']
```


Try it yourself

Try solving this question here:

 Java

 Python3

 JS


 C++

```
1 def fruits_into_baskets(fruits):
2     # TODO: Write your code here
3     return -1
4
```

Test

Save

Reset



Solution

This problem follows the **Sliding Window** pattern and is quite similar to [Longest Substring with K Distinct Characters](#). In this problem, we need to find the length of the longest subarray with no more than two distinct characters (or fruit types!). This transforms the current problem into **Longest Substring with K Distinct**

Characters where $K=2$.

Code

Here is what our algorithm will look like, only the highlighted lines are different from [Longest Substring with K Distinct Characters](#):

```
1 def fruits_into_baskets(fruits):
2     window_start = 0
3     max_length = 0
4     fruit_frequency = {}
5
6     # try to extend the range [window_start, window_end]
7     for window_end in range(len(fruits)):
8         right_fruit = fruits[window_end]
9         if right_fruit not in fruit_frequency:
10             fruit_frequency[right_fruit] = 0
11             fruit_frequency[right_fruit] += 1
12
13     # shrink the sliding window, until we are left with '2' fruits in the fruit frequency dictionary
14     while len(fruit_frequency) > 2:
15         left_fruit = fruits[window_start]
16         fruit_frequency[left_fruit] -= 1
17         if fruit_frequency[left_fruit] == 0:
18             del fruit_frequency[left_fruit]
19         window_start += 1 # shrink the window
20     max_length = max(max_length, window_end - window_start + 1)
21     return max_length
22
23
24 def main():
25     print("Maximum number of fruits: " + str(fruits_into_baskets(['A', 'B', 'C', 'A', 'C'])))
26     print("Maximum number of fruits: " + str(fruits_into_baskets(['A', 'B', 'C', 'B', 'B', 'C'])))
27
28
29 main()
30
```

Time Complexity

The above algorithm's time complexity will be $O(N)$, where 'N' is the number of characters in the input array. The outer **for** loop runs for all characters, and the inner **while** loop processes each character only once; therefore, the time complexity of the algorithm will be $O(N + N)$, which is asymptotically equivalent to $O(N)$.

Space Complexity

The algorithm runs in constant space $O(1)$ as there can be a maximum of three types of fruits stored in the frequency map.

Similar Problems

Problem 1: Longest Substring with at most 2 distinct characters

Given a string, find the length of the longest substring in it with at most two distinct characters.

Solution: This problem is exactly similar to our parent problem.

[← Back](#)

[Next →](#)

Longest Substring with K Distinct Cha...

No-repeat Substring (hard)

✓ Completed

