904. Fruit Into Baskets

Medium · Topics · Companies

You are visiting a farm that has a single row of fruit trees arranged from left to right. The trees are represented by an integer array `fruits` where `fruits[i]` is the **type** of fruit the i^{th} tree produces.

You want to collect as much fruit as possible. However, the owner has some strict rules that you must follow:

- You only have **two** baskets, and each basket can only hold a **single type** of fruit. There is no limit on the amount of fruit each basket can hold.
- Starting from any tree of your choice, you must pick **exactly one fruit** from **every** tree (including the start tree) while moving to the right. The picked fruits must fit in one of your baskets.
- Once you reach a tree with fruit that cannot fit in your baskets, you must stop.

Given the integer array `fruits`, return *the **maximum** number of fruits you can pick*.

Example 1:

```
Input: fruits = [1,2,1]
Output: 3
Explanation: We can pick from all 3 trees.
```

Example 2:

```
Input: fruits = [0,1,2,2]
Output: 3
Explanation: We can pick from trees [1,2,2].
If we had started at the first tree, we would only pick from trees [0,1].
```

Example 3:

```
Input: fruits = [1,2,3,2,2]
Output: 4
Explanation: We can pick from trees [2,3,2,2].
If we had started at the first tree, we would only pick from trees [1,2].
```

Constraints:

- `1 <= fruits.length <= 10^5`
- `0 <= fruits[i] < fruits.length`

Approach 1: Brute force with the help of hashset

```
for( i : 0 to n-1)
{
        set.clear( )
    for( j : i to n-1)
    {
        push a_j into set
        if ( set.size > 2)  break
```

else

$ans = max(ans, j-i+1)$

}

}

$T(n) : O(n^2)$
$S(n) : O(2)$

Approach 2: Sliding window + hash map

here you can't use hashset bcoz we can't keep track of crct window if we delete an element from hash set.

i.e. $\{ \boxed{3 \ 3 \ 3 \ 1 \ 1 \ 2} \ 4 \ 5 \}$

$\quad l \qquad\qquad\qquad r$

here we found that $w_s > 2$. So now we need to shrink the window size to 2 by removing element pointing by l pointer. If we remove 3 from set, set size becomes 2 but l pointer still points to the next occurrence of 3 in window which is wrong window.

So we make use of hashmap to store frequency also of an element and implement sliding window approach crctly.

$l = 0, r = 0$

while $(r < n)$

{

$m[a_r]$++

```
while( l < n && m.size > 2)
{
      m[a_l]--
     if( m[a_l] == 0)    m.erase(a_l)
      l++
}

ans = max(ans, r-l+1)
r++
}
```

$T(n) : O(n)$
$S(n) : O(2)$