229. Majority Element II

27 June 2024 Thursday
Kaushal Pratap

In this problem, we have to find the elements which appear more than *n/3* time in an array *nums* where *n* is the length of the array. We should return the output of resulting elements in the form of a list.

For example -

Input: nums = [3, 2, 3]
Output: [3]

Here, 3 repeats more than 1 time. Because *n//3* is 1. Floor division is not necessary because 3 repeats 2 time and it's greater then 1.5.

Logic-

We will first initialize an empty Hash Table and start traversing through our array and access each element. If the element is not in Hash Table, i.e. it's appearing for the first time, then we will add it to the Hash Table with its quantity 1. And if the element is already present in the Hash Table, then we will add that element with the addition of 1 to its previous quantity as it appeared one more time.

For example-

Nums = [3, 2, 3]
HashMap = {}
Now, we will access three and it's not in Hash Table, so we will add it to the table to its quantity 1.
I.e. {3:1}
Now for two, our Hash Table looks like this - {3:1, 2:1}
And finally for three, it'll look like this - {3:2, 2:1}

We got our Hash Table ready. Now, we will traverse through our Hash Table and access the *values* to their particular *key* i.e. the repetition quantity of each element. And if the quantity is greater than *n/3,* we will add that element to our array which we will return at the end.

Let us now look at the code for better understanding.

Code-

```
5  class Solution:
6      def majorityElement(self, nums: list[int]) -> list[int]:
```

- We will begin by creating our class and function in line 5 and 6.
- Our function will take *nums* as list input.

```
7          HashMap = {}
8          array = []
```

- Then, we will initialize our empty Hash Table and array in which we will append our elements in line 7 and 8 respectively.

```
9          for i in nums:
10             try:
11                 HashMap[i] = HashMap[i] + 1
12             except:
13                 HashMap[i] = 1
```

- We will start traversing through our array *nums* using for loop in line 9.
- In line 11, we will check if the element already exists in Hash Table. If it does exist, then we will increment its quantity by 1 in line 13.
- And if it doesn't exist, we will simply add the element with its quantity 1 in Hash Table.
- But the catch is here about the *try* and *except* statements in line 10 and 12.
- *Try* and *except* behave like *if* and *else* statement. But if we used traditional *if* statement, it will always raise an error in line 11 if an element is occurring for the first time and will put our program on a halt. So, it's better to use *try* statement as it handles the error. And the *except* statement is executed when there's an error in executing *try* statement.
- After the end of the loop, we'll have our Hash Table ready with elements mapped to their particular quantity.

```
15              for j in HashMap:
16                  if HashMap[j] > len(nums)/3:
17                      array.append(j)
```

- Now in line 15, we will start traversing through our Hash Table using a for loop.
- In line 16, we will check if the quantity of a particular element in Hash Table is more than *n/3*.
- If yes, we will append that element to our array in line 17.
- At the end of the loop, we will have our array ready with the elements which occur more than *n/3 times*.

```
19              return array
```

- Finally, we will return our array in line 19.

Time complexity analysis-

1. Creating class, function, Hash Table and array in line 5,6,7 and 8 respectively have O(1) time complexity.
2. In line 9, for loop have a time complexity of O(n) as it traverses through n number of elements.
3. And all operations inside for loop from line 10 to 13, accounts for O(1). So, the resulting time complexity of this loop will be O(n).
4. From line 15 to 17, this for loop also have a time complexity of O(n).
5. And in the end, returning our array in line 19 also accounts for O(1).

So, the overall time complexity of our algorithm is O(n +n) = O(2n) which is ***O(n).***

Leetcode - https://leetcode.com/problems/majority-element-ii/
GitHub - https://github.com/kaushal-pratap/code-portifolio/blob/main/Leetcode%20Problems/229.%20Majority%20Element%20II.py

```python
class Solution:
    def majorityElement(self, nums: list[int]) -> list[int]:
        HashMap = {}
        array = []
        for i in nums:
            try:
                HashMap[i] = HashMap[i] + 1
            except:
                HashMap[i] = 1

        for j in HashMap:
            if HashMap[j] > len(nums)/3:
                array.append(j)

        return array
```