October 26, 2021 ▪ Arrays / Data Structure

Find the repeating and missing numbers

Problem Statement: You are given a read-only array of N integers with values also in the range [1, N] both inclusive. Each integer appears exactly once except A which appears twice and B which is missing. The task is to find the repeating and missing numbers A and B where A repeats twice and B is missing.

Example 1:

```
Input Format:  array[] = {3,1,2,5,3}

Result: {3,4)

Explanation: A = 3 , B = 4
Since 3 is appearing twice and 4 is missing
```

Example 2:

```
Input Format: array[] = {3,1,2,5,4,6,7,5}

Result: {5,8)

Explanation: A = 5 , B = 8
Since 5 is appearing twice and 8 is missing
```

Solution

Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

Solution 1: Using Count Sort

Intuition + Approach:

Since the numbers are from 1 to N in the array arr[]

1. Take a substitute array of size N+1 and initalize it with 0.
2. Traverse the given array and increase the value of substitute[arr[i]] by one .
3. Then again traverse the substitute array starting from index 1 to N.

If you find any index value greater than 1 that is repeating element A.

If you find any index value = 0 then that is the missing element B.

Dry Run: lets take the example of array[] = {3,1,2,5,3}

The size of the array is 5

We initialize a substitute array of size 6 with elements 0.

Now traversing through the array

1. Found 3 at 0 indexes, increase the value of substitute array at index 3 by 1.
2. Found 1 at 1 index, increase the value of substitute array at index 1 by 1.
3. Found 2 at 2 indexes, increase the value of substitute array at index 2 by 1.
4. Found 5 at 3 indexes, increase the value of substitute array at index 5 by 1.
5. Found 3 at 4 indexes, increase the value of substitute array at index 3 by 1.

Now Traversing through the substitute array

At index 3, the value is greater than 1 i.e 2. So A = 3.

At index 4, the value is 0 so B = 4.

Code:

C++ Code

```
vector<int> find_missing_repeating(vector<int> array)
{
    int n = array.size() + 1;

    vector<int> substitute(n, 0); // initializing the substitute array with 0 of size n+1.

    vector<int> ans;              // initializing the answer  array .

    for (int i = 0; i < array.size(); i++)
    {
        substitute[array[i]]++;
    }

    for (int i = 1; i <= array.size(); i++)
    {
        if (substitute[i] == 0 || substitute[i] > 1)
        {
            ans.push_back(i);
        }
    }

    return ans;
}
```

Java Code

```
static ArrayList<Integer> find_missing_repeating(List<Integer> array) {
        int n = array.size() + 1;

        int[] substitute = new int[n]; // initializing the substitute array with 0 of size n+1.
        for (int i = 0; i < n; i++)
            substitute[i] = 0;

        ArrayList<Integer> ans = new ArrayList<>();              // initializing the answer  array .

        for (int i = 0; i < array.size(); i++) {
            substitute[array.get(i)]++;
        }

        for (int i = 1; i <= array.size(); i++) {
            if (substitute[i] == 0 || substitute[i] > 1) {
                ans.add(i);
            }
        }

        return ans;
    }
```

Python Code

```
from typing import List
def find_missing_repeating(array: List[int]) -> List[int]:
    n = len(array) + 1


    # initializing the substitute array with 0 of size n+1.
    substitute = [0] * n


    ans = []  # initializing the answer  array .


    for i in range(len(array)):
        substitute[array[i]] += 1


    for i in range(1, len(array) + 1):
        if substitute[i] == 0 or substitute[i] > 1:
            ans.append(i)


    return ans
```

Time Complexity: O(N)

Space Complexity: O(N) As we are making new substitute array

Solution 2: Using Maths

Intuition + Approach:

The idea is to convert the given problem into mathematical equations. Since we have two variables where one is missing and one is repeating, can we form two linear equations and then solve for the values of these two variables using the equations?

Let’s see how.

Assume the missing number to be X and the repeating one to be Y.

Now since the numbers are from 1 to N in the array arr[]. Let's calculate sum of all integers from 1 to N and sum of squares of all integers from 1 to N. These can easily be done using mathematical formulae.

Therefore,

Sum of all elements from 1 to N:

```
S = N*(N+1)/2 ---- equation 1
```

And, Sum of squares of all elements from 1 to N:

```
P = N(N+1)(2N+1)/6. ----- equation 2
```

Similarly, find the sum of all elements of the array and the sum of squares of all elements of the array respectively.

- s1 = Sum of all elements of the array. —– equation 3
- P1 = Sum of squares of all elements of the array. ———– equation 4

Now, if we subtract the sum of all elements of the array from the sum of all elements from 1 to N, that should give us the value for (X – Y).

Therefore,

(X-Y) = s – s1 = s'

Similarly,

X^2 – Y^2 = P – P1 = P'

or, (X + Y)(X – Y) = P'

or, (X + Y)*s' = P'

or, X + Y = P'/s'

Great,

we have the two equations we need:

(X – Y) = s'

(X + Y) = P'/s'

We can use the two equations to solve and find values for X and Y respectively.

Note: here s and P can be large so take a long long int the data type.

Code:

C++ Code

```cpp
vector<int>missing_repeated_number(const vector<int> &A) {
    long long int len = A.size();

    long long int S = (len * (len+1) ) /2;
    long long int P = (len * (len +1) *(2*len +1) )/6;
    long long int missingNumber=0, repeating=0;

    for(int i=0;i<A.size(); i++){
       S -= (long long int)A[i];
       P -= (long long int)A[i]*(long long int)A[i];
    }

    missingNumber = (S + P/S)/2;

    repeating = missingNumber - S;

    vector <int> ans;

    ans.push_back(repeating);
    ans.push_back(missingNumber);

    return ans;
}
```

Java Code

```java
static ArrayList<Integer> missing_repeated_number(List<Integer> A) {
        long len = A.size();

        long S = (len * (len + 1) ) / 2;
        long P = (len * (len + 1) * (2 * len + 1) ) / 6;
        long missingNumber = 0, repeating = 0;
```

```
        for (int i = 0; i < A.size(); i++) {
            S -= (long)A.get(i);
            P -= (long)A.get(i) * (long)A.get(i);
        }

        missingNumber = (S + P / S) / 2;

        repeating = missingNumber - S;

        ArrayList<Integer> ans = new ArrayList<>();
```



```
        ans.add((int)missingNumber);

        return ans;
    }
```

Python Code

```
from typing import List
def find_missing_repeating(arr: List[int]) -> List[int]:
    n = len(arr)
    s = (n * (n + 1)) // 2
    p = (n * (n + 1) * (2 * n + 1)) // 6
    missing_number = 0
    repeating = 0


    for i in range(n):
        s -= arr[i]
        p -= arr[i] * arr[i]


    missing_number = (s + p // s) // 2
    repeating = missing_number - s


    ans = [repeating, missing_number]
    return ans
```

Time Complexity: O(N)

Space Complexity: O(1) As we are NOT USING EXTRA SPACE

Solution 3: XOR

Intuition + Approach:

Let x and y be the desired output elements.

Calculate the XOR of all the array elements.

xor1 = arr[0]^arr[1]^arr[2].....arr[n-1]

XOR the result with all numbers from 1 to n

xor1 = xor1^1^2^.....^n

xor1 will have the result as (x^y), as others would get canceled. Since we are doing XOR, we know xor of 1 and 0, is only 1, so all the set bits in xor1, mean that the index bit is only set at x or y.

So we can take any set bit, in code we have taken the rightmost set bit, iterate over it, and divide the numbers into two hypothetical buckets.

If we check for numbers with that particular index bit set, we will get a set of numbers that belongs to the first bucket, also we will get another set of numbers belonging to the second bucket. The first bucket will be containing either x or y, similarly, the second bucket will also be containing either x or y. XOR of all elements in the first bucket will give X or Y, and XOR of all elements of the second bucket will give either X or Y since there will be double instances of every number in each bucket except the X or Y.

We just need to iterate again to check which one is X, and which one is y. Can be simply checked by linear iterations. For a better understanding, you can check the video explanation.

Code:

C++ Code

```
vector < int >Solution::repeatedNumber (const vector < int >&arr) {
    /* Will hold xor of all elements and numbers from 1 to n */
    int xor1;

    /* Will have only single set bit of xor1 */
    int set_bit_no;

    int i;
    int x = 0; // missing
    int y = 0; // repeated
```

```
    int n = arr.size();

    xor1 = arr[0];

    /* Get the xor of all array elements */
    for (i = 1; i < n; i++)
        xor1 = xor1 ^ arr[i];

    /* XOR the previous result with numbers from 1 to n */
    for (i = 1; i <= n; i++)
        xor1 = xor1 ^ i;

    /* Get the rightmost set bit in set_bit_no */
    set_bit_no = xor1 & ~(xor1 - 1);

    /* Now divide elements into two sets by comparing a rightmost set bit
       of xor1 with the bit at the same position in each element.
       Also, get XORs of two sets. The two XORs are the output elements.
       The following two for loops serve the purpose */

    for (i = 0; i < n; i++) {
        if (arr[i] & set_bit_no)
            /* arr[i] belongs to first set */
            x = x ^ arr[i];

        else
            /* arr[i] belongs to second set */
            y = y ^ arr[i];
    }

    for (i = 1; i <= n; i++) {
        if (i & set_bit_no)
            /* i belongs to first set */
            x = x ^ i;

        else
            /* i belongs to second set */
            y = y ^ i;
    }

    // NB! numbers can be swapped, maybe do a check ?
    int x_count = 0;
    for (int i=0; i<n; i++) {
        if (arr[i]==x)
            x_count++;
    }

    if (x_count==0)
        return {y, x};

    return {x, y};
}
```

Java Code

```
static int x, y;

    static void getTwoElements(int arr[], int n)
    {
        /* Will hold xor of all elements
       and numbers from 1 to n  */
        int xor1;

        /* Will have only single set bit of xor1 */
        int set_bit_no;

        int i;
        x = 0;
        y = 0;

        xor1 = arr[0];

        /* Get the xor of all array elements  */
        for (i = 1; i < n; i++)
            xor1 = xor1 ^ arr[i];

        /* XOR the previous result with numbers from
       1 to n*/
        for (i = 1; i <= n; i++)
            xor1 = xor1 ^ i;

        /* Get the rightmost set bit in set_bit_no */
        set_bit_no = xor1 & ~(xor1 - 1);

        /* Now divide elements into two sets by comparing
    rightmost set bit of xor1 with the bit at the same
    position in each element. Also, get XORs of two
    sets. The two XORs are the output elements. The
    following two for loops serve the purpose */
        for (i = 0; i < n; i++) {
            if ((arr[i] & set_bit_no) != 0)
                /* arr[i] belongs to first set */
                x = x ^ arr[i];
```

```
            else
                /* arr[i] belongs to second set*/
                y = y ^ arr[i];
        }
        for (i = 1; i <= n; i++) {
            if ((i & set_bit_no) != 0)
                /* i belongs to first set */
                x = x ^ i;

            else
                /* i belongs to second set*/
                y = y ^ i;
        }

        // at last do a linear check which amont x and y is missing or repeating

        /* *x and *y hold the desired output elements */
    }
```

Python Code

```
from typing import List
def find_missing_repeating(arr: List[int]) -> List[int]:
    # Will hold xor of all elements and numbers from 1 to n
    xor1 = arr[0]


    # Will have only single set bit of xor1
    set_bit_no = 0


    # Get the xor of all array elements
    for i in range(1, len(arr)):
        xor1 = xor1 ^ arr[i]


    # XOR the previous result with numbers from 1 to n
    for i in range(1, len(arr) + 1):
        xor1 = xor1 ^ i


    # Get the rightmost set bit in set_bit_no
    set_bit_no = xor1 & ~(xor1 - 1)


    # Now divide elements into two sets by comparing a rightmost set bit
    # of xor1 with the bit at the same position in each element.
    # Also, get XORs of two sets. The two XORs are the output elements.
    # The following two for loops serve the purpose
    x = 0
    y = 0
    for i in range(len(arr)):
        if arr[i] & set_bit_no:
            # arr[i] belongs to first set
            x = x ^ arr[i]
        else:
            # arr[i] belongs to second set
            y = y ^ arr[i]


    for i in range(1, len(arr) + 1):
        if i & set_bit_no:
            # i belongs to first set
            x = x ^ i
        else:
            # i belongs to second set
            y = y ^ i


    # NB! numbers can be swapped, maybe do a check ?
    x_count = 0
    for i in range(len(arr)):
        if arr[i] == x:
            x_count += 1


    if x_count == 0:
        return [y, x]


    return [x, y]
```

Note: This method doesn't cause overflow, but it doesn't tell which one occurs twice and which one is missing. We can add one more step that checks which one is missing and which one is repeating by iterating over the array. This can be easily done in O(N) time.

Time Complexity: O(N)

Space Complexity: O(1) As we are NOT USING EXTRA SPACE

> Special thanks to **Ambuj Kumar** , **Sudip Ghosh** and **KRITIDIPTA GHOSH** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, please check out this article

Find the Missing and Repeating Number | GFG | C++ and Java | Brute-Better-Optimal-Optimal

« Previous Post
Length of the longest subarray with zero Sum

Next Post »
Maximum depth of a Binary Tree

Load Comments