Tóm tắt đề bài

Cho mảng gồm có $n * m + 1$ phần tử dương, trong đó n phần tử xuất hiện chính xác m lần (m khác 1), 1 phần tử xuất hiện chính xác 1 lần.
Yêu cầu: Tìm phần tử xuất hiện 1 lần.

Input mẫu

```
N = 3, M = 3
Arr = [2 2 5 5 3 3 5 7 3 2]
```

Output mẫu

```
7
```

Giải thích: 2, 5, 3 đều xuất hiện chính xác 3 lần, 7 chỉ xuất hiện 1 lần.

Lời giải

Cách giải:

Cách 1: Xử lý theo từng bit & xor.

Xét lần lượt từng bit, với mỗi bit ta xét tất cả các số trong Array, tăng số lần xuất hiện của bit thứ i nếu như
$(Arr_j \ \& \ 2^i) = 1.$
Kết quả sẽ là các vị trí bit xuất hiện khác m lần.

```
int FindUniqueNumber(int n, int m, vector<int>& arr){
    int result = 0;
    int x, sum;

    for (int i = 0; i < INT_SIZE; i++){
        sum = 0;
        x = (1 << i);
        for (int j = 0; j < arr.size(); j++){
            if (arr[j] & x) sum++;
        }
        if (sum % m != 0){
            result |= x;
        }
    }
    return result;
}
```

Time complexity: O(INT_SIZE x N) với N là số lượng phần tử của Array.
Auxiliary Space : O(1)

Cách 2: Giải phương trình & dùng set.

Gọi n phần tử xuất hiện m lần trong array là $a_1, a_2, ..., a_n$, phần tử còn lại là n.
Ta có:
$m * (a_1 + a_2 + .. + a_n + b) - sum(Array) = (m - 1) * b$
Tuy nhiên để tìm được tập các phần tử phần biệt $a_1, a_2, ..., a_n$ ta dùng thêm set, dict,...
Từ đó ta tính được kết quả $b = (m * sum\_set - sum(Array))/m - 1.$

```
int FindUniqueNumber(int n, int m, vector<int>& arr){
    unordered_set<int> s(arr.begin(), arr.end());
    long long arr_sum = 0;
    for (int element: arr){
        arr_sum += element;
    }
    long long set_sum = 0;
    unordered_set<int> :: iterator itr;
    for (itr = s.begin(); itr != s.end(); itr++)
        set_sum += *itr;
    return (1ll * m * set_sum - arr_sum) / (m - 1);
}
```

Time complexity: O(Nlog(N))
Auxiliary Space : O(N)

Cách 3: Xử lý Xor.

Cách này chỉ hoạt động với m = 3, em chưa nghĩ ra cách giải cho trường hợp tổng quát :(.
Ta gọi:

- ones là các bit đã xuất hiện với số lần % 3 = 1.
- twos là các bit đã xuất hiện với số lần % 3 = 2.
  Duyệt qua từng phần tử arr_i, ta có 3 trường hợp:
- arr_i xuất hiện 1 lần, ta lấy nó Xor với ones.
- arr_i xuất hiện 2 lần, nghĩa là arr_i đã xuất hiện 1 lần, ta lấy arr_i & ones
- arr_i xuất hiện 3 lần, ta sẽ lấy đi các bit của arr_i trong ones và twos.

Kết quả cuối cùng sẽ là ones, cách hoạt động như sau:

- Nếu arr_i xuất hiện lần đầu, `(ones & arr[i])` = 0 và twos không đổi. Ones ghi nhận các bit của arr_i: `ones ^ arr[i]`. Do arr_i chỉ xuất hiện trong ones, nên ones và twos sẽ không có bit chung, common_bit không đổi và việc remove bits từ ones và twos không xảy ra.
- Nếu arr_i xuất hiện lần 2, ones sẽ có các bits của arr_i, two sẽ được ghi nhận `twos = twos | (ones & arr[i])`, và do nó xuất hiện 2 lần nên ta remove các bits của x trong ones bằng `ones = ones ^ arr[i]` (vì x^x = 0). Và tương tự như trường hợp trên, ones và twos sẽ không có bit chung, common_bit không đổi.
- Nếu arr_i xuất hiện lần 3, ones chưa có các bits của arr_i nhưng $twos$ thì có. Sau đó ghi nhận các bits của arr_i trong ones: `ones = ones ^ arr[i]`. Bây giờ cả ones và twos đều có các bits của arr_i, common_bit_mask sẽ lưu lại các bits chung, và remove khỏi ones và twos.

```
int FindUniqueNumber(int n, int m, vector<int>& arr){
    int ones = 0, twos = 0;

    int common_bit_mask;
    for (int i = 0; i < arr.size(); i++) {

        // "ones & arr[i]" gives the bits in both ones & arr[i]
        twos = twos | (ones & arr[i]);

        // xor new bits with previous'ones'
        // with bit appearing odd time
        ones = ones ^ arr[i];

        // The common_bit_mask are those bits appear
        // third time, it shouldn't be in "ones" and "twos"
        common_bit_mask = ~(ones & twos);

        // Remove common bits in ones & twos
        ones &= common_bit_mask;
        twos &= common_bit_mask;
    }

    return ones;
}
```

Em nghĩ có thể mở rộng với bài toán xuất hiện m lần, nhưng hiện tại em vẫn chưa nghĩ ra cách làm.
Time complexity O(n)
Auxiliary Space : O(1)