Count the number of subarrays with given xor K

Problem Statement: Given an array of integers A and an integer B. Find the total number of subarrays having bitwise XOR of all elements equal to B.

Examples:

Input Format: A = [4, 2, 2, 6, 4] , B = 6

Result: 4

Explanation: The subarrays having XOR of their elements as 6 are [4, 2], [4, 2, 2, 6, 4], [2, 2, 6], [6]

Input Format: A = [5, 6, 7, 8, 9], B = 5

Result: 2

Explanation: The subarrays having XOR of their elements as 2 are [5] and [5, 6, 7, 8, 9]

Solution:

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

Solution 1: Brute Force

Intuition: The brute force solution is to generate all possible subarrays. For each generated subarray we get the respective XOR and then check if this XOR is equal to B. If it is then we increment the count. In the end, we will get the count of all possible subarrays that have XOR equal to B.

Approach:

1. Generate subarrays: To generate all possible subarrays, we use the same old technique of nested loops. For each value of the outer loop (i loop), the inner loop(j loop) runs from i till the last element. Each iteration of the inner loop gives a new subarray.
2. Maintain XOR: Since each iteration of the inner loop gives a new subarray, we maintain a variable X, in which we keep the XOR of the current subarray.
3. Check and Count: Before moving to the next iteration of the inner loop (that is before going to the next subarray), we check if the current XOR is equal to B, if it is then we increment the counter (counter also has to be maintained).

```
public int solve(int[] A, int B) {
    int c=0;
    for(int i=0;i<A.length;i++){
       int current_xor = 0;
       for(int j=i;j<A.length;j++){
          current_xor^=A[j];
          if(current_xor==B) c++;
       }
    }
    return c;
  }
```

Time Complexity: $O(N^2)$

Space Complexity: O(1)

Solution 2: Prefix xor and map

Intuition: The main idea is to observe the prefix xor of the array. Prefix Xor is just another array, where each index contains XOR of all elements of the original array starting from index 0 up to that index. In other words
prefix_xor[i] = XOR(a[0], a[1], a[2],……,a[I])

Once we have made the prefix xor array, we observe a fact:

P = xor(a[0], a[1], a[2],……., a[q], a[q+1],….., a[p])
Q = xor(a[0], a[1], a[2],……., a[q])

Therefore,
P^Q = xor(a[q+1],….., a[p]) = M
So, now we understand that from the prefix xor array when we XOR two elements at different indices we get the xor of the elements (in the original array) that were between those indices.

Let's say we did XOR(P, B) and we got Q (B is the integer given in question). What does this mean?
This means that the subarray between q and p is having xor = B. To understand this we just use simple equations:
P^B = Q
=> P^B^P = Q^P
=> B = Q^P

And we already know by fact 1 that Q^P will give xor of all elements between q and p. Therefore, the subarray between q and p has xor = B.
Suppose we did XOR(P, B) and we got Q (B is the integer given in question). But there is more than one Q before p.

In this case, there are two subarrays that have XOR = B. Subarrays between q1 to p and q2 to p.

IMP NOTE: although we talk about prefix xor "array", it should be noted that at a time we need only one element of this array. Hence, we can just use a variable to maintain the prefix xor.

Approach: We need to traverse the array while we maintain variables for current_perfix_xor, counter, and also a map to keep track of visited xors. This map will maintain the frequency count of all previous visited current_prefix_xor values. If for any index current_prefix_xor is equal to B, we increment the counter. If for any index we find that Z is present in the visited map, we increment the counter by visited[Z] (Z=current_prefi_xor^B). At every index, we insert the current_prefix_xor into the visited map with its frequency.

```
public int solve(int[] A, int B) {
    HashMap<Integer,Integer> visitedMap =new HashMap<Integer,Integer> ();
    int count=0;
    int xor=0;
    int n=A.length;
    for(int i=0;i<n;i++)
    {
       xor=xor^A[i];
```

```
        if(xor==B)
        count++;
        if(visitedMap.containsKey(xor^B))
        {
           count+=visitedMap.get(xor^B);
        }
        visitedMap.put(xor,visitedMap.getOrDefault(xor,0)+1);
    }
    return count;
}
```

Time Complexity: O(N)

Space Complexity: O(N)