Count Subarray with xor k

BRUTE FORCE

xor every possible subarray and count whichever gives k

```
#include <bits/stdc++.h>

int subarraysXor(vector<int> &arr, int x)
{
    //    Write your code here.
    int n=arr.size(),xr,cnt=0;
    for(int i=0;i<n;i++)
    {
        xr=0;
        for(int j=i;j<n;j++)
        {
            xr=xr^arr[j];
            if(xr==x)
                cnt++;
        }
    }
    return cnt;
}
```

- Time Complexity : O(N^2)
- Space Complexity : O(1)

Optimised Approach :

Just as in subarray sum with sum k we used prefix sum we will use the same strategy we will store the previous xor counts in the map and if the xor is fount in map we will add to out count.

as for subarray sum we were finding x-k in the prefix sum and if it exists we know that k will exist Similary here we find xr^k and if it exists k will exist so we will add it to out ans and return the final count after traversing whole subarray.

Intuition is to find whether we have encountered k previously in the given subaray or not and if encountered it will be store in map as we store all the xor values count in map.

How to derive we need to find xr^k in map?

we consider we have x and k xor stored in xr variable the we can write it as :

x^k=xr

now xoring by k on both sides we get

x=xr^k as doing xor with same variable cancels each other.

```
#include <bits/stdc++.h>

int subarraysXor(vector<int> &arr, int x)
{
    //    Write your code here.
    unordered_map<int,int> m;
    m[0]=1;
    int cnt=0,xr=0,n=arr.size();
    for(int i=0;i<n;i++)
    {
        xr=xr^arr[i];
        if(m.find(xr^x)!=m.end())
        {
            cnt+=m[xr^x];
        }
        m[xr]++;

    }
    return cnt;
}
```

- Time Complexity : O(N)
- Space Complexity : O(N)