

DAA ASSIGNMENT 5

Group No 19

AYUSH KHANDELWAL (IIT2019240)

AYUSH BHAGTA (IIT2019501)

TAUHID ALAM (BIM2015003)

Contents -

- Problem Statement
- Introduction
- Proposed Algorithm
- Illustration
- Time and space complexity analysis
- Conclusion
- References

Problem Statement

In this report we designed a Dynamic Programming algorithm to find the number of subsets of given array of n elements `arr[]` having XOR of elements as a given number K .

Introduction

Given an array `arr[]` of size `n`, we will use dynamic programming approach to find the number of elements having XOR value as `K`. Here in `dp[i][j]` we keep a count of number of sets from `0` to `i-1` having XOR value as `j`. At the end the program we will give `dp[n][k]` as output.

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of sub-problems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

ALGORITHMIC DESIGN

Following is Dynamic Programming algorithm

- We initialize all values of $dp[i][j]$ as 0.
- Set value of $dp[0][0] = 1$ since XOR of an empty set is 0.
- Iterate over all the values of $arr[i]$ from left to right and for each $arr[i]$, iterate over all the possible values of XOR i.e. from 0 to m (both inclusive).

Here m is maximum possible value for XOR of any of possible subsets. To calculate m , we take maximum element from array and

$$m = (1 \ll (\text{int})(\log_2(\text{max}) + 1)) - 1;$$

- Fill the dp array as following:

for i = 1 to n:

for j = 0 to m:

$$dp[i][j] = dp[i - 1][j] + dp[i - 1][j \oplus arr[i - 1]]$$

- This can be explained as, if there is a subset $arr[0 \dots i-2]$ with XOR value j , then there also exists a subset $arr[0 \dots i-1]$ with XOR value j also if there exists a subset $arr[0 \dots i-2]$ with XOR value $j \oplus arr[i]$ then clearly there exist a subset $arr[0 \dots i-1]$ with XOR value j , as:

$$j \oplus arr[i - 1] \oplus arr[i - 1] = j.$$

- Counting the number of subsets with XOR value k : Since $dp[i][j]$ is the number of subsets having j as XOR value from the subsets of $arr[0..i-1]$, then the number of subsets from set $arr[0..n]$ having XOR value as K will be $dp[n][K]$

Illustration

Let's take a array $arr = [1, 2, 3, 4]$ and $K=6$

Firstly we initialize dp array as 0.

$dp[i][j] = 0$ (for all $i < n$ & $j < n$)

here in $dp[i][j]$ we keep a count of number of subsets from 0 to $i-1$ having XOR value as j

$n = \text{sizeof}(arr)$

$dp[0][0] = 1$ (Empty set)

Now take max element from array and use it to calculate maximum possible XOR value(m).

$max=4$

$m = (1 \ll (\text{int})(\log(4)+1)) - 1$

$m = 7$

Now we Iterate over all values of $arr[i]$ from $i=1$ to $i=n-1$. Then

for each iteration ,we also iterate over all values of XOR.

dp(After 1st Iteration):

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

dp(After 2nd Iteration):

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

dp(After 3rd Iteration):

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

dp(After 4th Iteration):

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Now our solution is the value of $dp[n][k] = dp[4][5] = 2$.

Time complexity

In this approach we iterate over whole array one by one finding and storing the possible subsets that generate a value p we say and store it in the $dp[i][k]$ which will need time to iterate over the array and for each element a loop of time complexity m , where m is the maximum possible value of XOR and could be found from the max value of element in array.

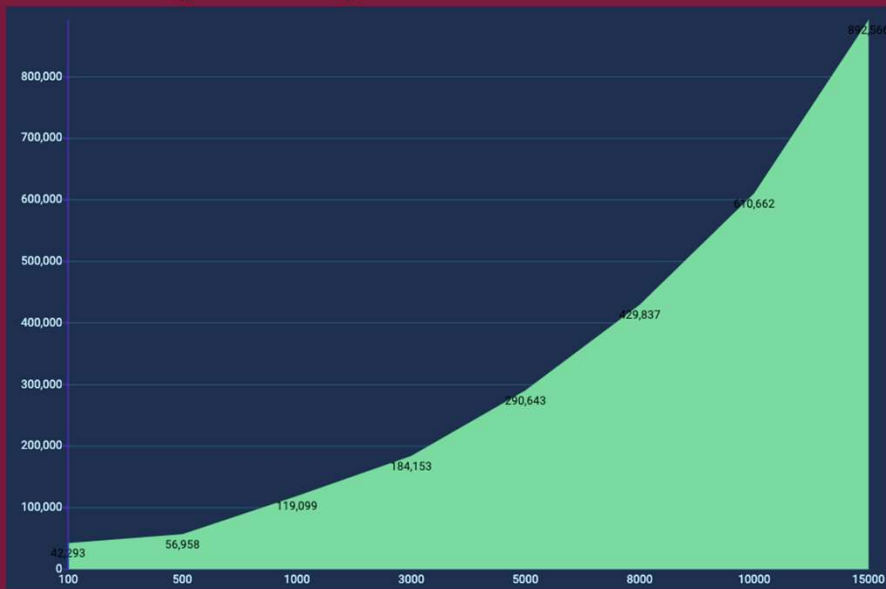
$$m = 2^{\lceil \log_2(\max\text{-element}) \rceil} - 1$$

This will result in the time complexity of $O(n * m)$.

N	M	Time(in nanoseconds)
10	100	31042
10	500	63804
10	1000	71490
10	3000	116213
10	5000	204245
10	8000	368717
10	10000	628989
10	15000	912566

N	M	Time(in nanoseconds)
100	10	42293
500	10	56958
1000	10	119099
3000	10	184153
5000	10	290643
8000	10	429837
10000	10	610662
15000	10	892566

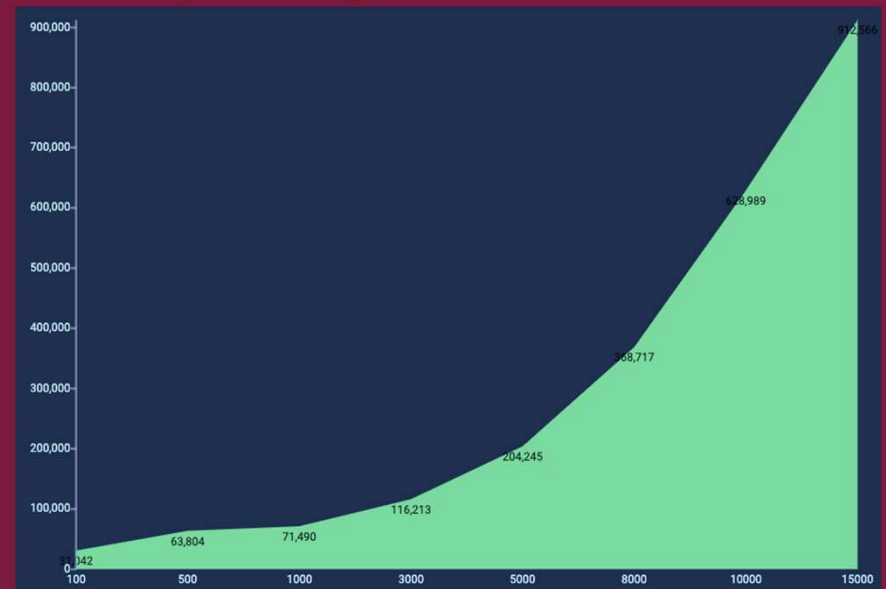
Graph of N Against time in nano seconds



X-axis: N

Y-axis: Time in ns

Graph of M Against time in nano seconds



X-axis: M

Y-axis: Time in ns

Space complexity

Brute-Force :

In Brute force we will be directly doing XOR operation on the variable or will be resetting the variable, no extra space other than input would be required . This will result in the space complexity of $O(1)$.

Dynamic-Programming :

In this approach we will have to store the values for all possible cases that are generated, as the $dp[i][j]$ requires $dp[i - 1][j]$ and hence a 2D array of size $n * m$ is required where m is same as mentioned above, that is:

$$m = 2[\log_2(\text{max-element})] + 1 - 1$$

This will result in the space complexity of $O(n * m)$.

Conclusion

We can observe that In Dynamic Programming Approach it may consume more space but will have better time complexity than Brute force approach.

References

- 1. <https://www.geeksforgeeks.org/calculate-xor-1-n/>
- 2. <https://www.geeksforgeeks.org/dynamic-programming/>
- 3. Cormen, Leiserson, Rivest, and Stein (2009). Introduction to Algorithms, 3rd edition.