Design and Analysis of Algorithms Assignment - 2

Department of Information Technology

Indian Institute of Information Technology - Allahabad, India

Jaidev Das
IIT2019197

Deeptarshi Biswas
IIT2019195

Abstract: ***In this paper we have approached the problem of finding, given an array of n numbers and a number K, the number of subsets of the array such that the XOR of its elements is equal to K. The XOR bitwise operator takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different. This paper shows how this problem can be solved using dynamic programming. First, we go over the stepwise algorithm and then follows the pseudocode of the solution code. At the end we go over the complexity analysis of the solution and see if it is efficient.***

Index Terms:Arrays, XOR bitwise operator, Binary numbers, Dynamic programming

INTRODUCTION

XOR Bitwise Operator The XOR bitwise operator takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.

Dynamic Programming 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 subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial. For example, if we write simple recursive solution for Fibonacci Numbers, we get exponential time complexity and if we optimize it by storing solutions of subproblems, time complexity reduces to linear.

Characteristics of Dynamic Programming

1. **Overlapping Subproblems:** Subproblems are smaller versions of the original problem. Any problem has overlapping sub-problems if finding its solution involves solving the same subproblem multiple times.
2. **Optimal Substructure Property:** Any problem has optimal substructure property if its overall optimal solution can be constructed from the optimal solutions of its subproblems.

This report further contains:

- Algorithm Designs
- Algorithm Analysis
- Conclusion
- References
- Appendix

ALGORITHM DESIGN

Naive approach: First we look at a naive approach to solve given problem. Here we are using brute force and backtracking. Pseucode for this approah is given below-

```
#include <iostream>
#include<vector>
using namespace std;
#define vec vector<int>
int n,k,cnt=0;
void subsets(int * a,vec& v,int ind)
{
    if(ind==n)
    {
        int xorr=0;
        for(auto it:v)
        xorr=xorr^it;
        if(xorr==k)
        cnt++;
        return;
    }
    subsets(a,v,ind+1);
    v.push_back(a[ind]);
    subsets(a,v,ind+1);
    v.pop_back();
}
int main()
{
   cin>>n>>k;
   int a[n];
   for(int i=0;i<n;i++)
   cin>>a[i];
   vec v;
   subsets(a,v,0);
   cout<<cnt;
```

```
    return 0;
}
```

Time complexity of this algorithm is $O((2^n) * n)$

Dynamic Programming approach: Basically, the approach based on Dynamic Programming has the steps defined in following algorithmic procedure, which have been implemented while solving the problem:

1. We define a number m such that m = m OR a[i] for all i from 1 to n. This number is actually the maximum value any XOR subset will acquire.

2. We create a 2D array dp[n+1][m+1], such that dp[i][j] equals the number of subsets having XOR value j from subsets of arr[0...i-1].

3. We fill the dp array as following: 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) and 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 XOR arr[i-1]] 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]

```
BEGIN:
        Take n and k input
        a[n+1]
        orr=0;
        FOR i 1 to n :
      INPUT(a[i])
      orr=orr|a[i]

        IF orr<k:
        PRINT 0

    dp[n+1][orr+1]
    INITIALIZE(dp to 0)

    dp[0][0]=1;

        FOR i 1 to n:
        FOR j 0 to orr:
            dp[i][j]=dp[i-1][j]+dp[i-1][orr^a[i
                ]]

    PRINT(dp[n][k])
END:
```

ALGORITHM ANALYSIS

Time complexity: Here, we are going through a matrix of order n times r, where n = number of elements in array r = OR of all elements in array

Hence time complexity comes out to be O(nr)

Space Complexity: Since we are making a matrix of order n times r, Space complexity is O(nr)

CONCLUSION

The method discussed above shows the mentioned problem can be solved efficiently for values upto 10^4 of both n and r using this solution based on dynamic programming. Here, n = number of elements in array r = OR of all elements in array

REFERENCES

1. Dynamic Programming: https://www.geeksforgeeks.org/dynamic-programming/
2. Bitwise operators in C and C++ https://www.geeksforgeeks.org/bitwise-operators-in-c-cpp
3. Understanding time complexity https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/

APPENDIX

To run the code, follow the following procedure:

1. Download the code(or project zip file) from the github repository.
2. Extract the zip file downloaded above.
3. Open the code with any IDE like Sublime Text, VS Code, Atom or some online compilers like GDB.
4. Run the code following the proper running commands(vary from IDE to IDE)
   (a) **For VS Code:** Press Function+F6 key and provide the input on the terminal.
   (b) **For Sublime Text:** Click on the Run button and provide the input.

Code for Implementation is:

```
#include<bits/stdc++.h>
using namespace std;
#include<vector>
#include<map>
#include<queue>
#include<utility>
#include<set>
#define lli long long int
#define maplli map<lli,lli>
#define iterm maplli::iterator
#define fi first
#define si second
#define For(i,a,b) for(lli i=a;i<b;i++)
#define vec vector<lli>
#define pb push_back
#define mp make_pair
```

```
#define iterv vector<lli>::iterator
#define pi pair<lli,lli>
#define vecpi vector<pi>
#define iterpi vector<pi>::iterator
#define Q queue
#define PQ priority_queue
#define mapc map<char,lli>
#define itermc map<char,lli>::iterator
#define vecset vector<set<lli>>
#define itervs vecset::iteratorlli
#define Forb(i,a,b) for(lli i=a;i>=b;i--)
#include<tuple>
#define tal tuple<lli,lli,lli>
int main()
{
    lli t;
   cout<<"ENTER TOTAL NUMBER OF TEST CASES:"<<
       endl;
    cin>>t;
    cout<<"ENTER THE TEST CASES:"<<endl;
    while(t--)
    {
        lli n,k;
       cout<<"ENTER LENGTH OF ARRAY AND THE
           VALUE OF K:"<<endl;
        cin>>n>>k;
        lli K=k;
        lli a[n+1];
        lli orr=0;
        cout<<"ENTER ARRAY ELEMENTS:"<<endl;
        lli maxi=0;
        For(i,1,n+1){cin>>a[i];orr=orr|a[i];}
        if(orr<k)
        {
            cout<<"0"<<endl;continue;
        }
        k=orr;
        lli dp[n+1][k+1];
        memset(dp,0,sizeof(dp));
        dp[0][0]=1;
        For(i,1,n+1)
        {
            For(j,0,k+1)
            {
                if((j^a[i])<=k)
                        dp[i][j]=dp[i-1][j]+dp[i
                            -1][j^a[i]];
                else
                        dp[i][j]=dp[i-1][j];
            }
        }
        cout<<"Answer = "<<dp[n][K]<<endl<<endl;
    }
}
```