problem statement:

> Given an array arr of N integers and an integer K, find the number of subsets of arr having XOR of elements as K.

Example:

Input:

```
N = 4
k = 6
arr: 6 9 4 2
```

Output:

```
2
```

Explanation:
The subsets are
{4,2} and {6}

Approach:

⟶ Here we will generate all the subsets and here we have 2 choices:

```
1. Take the current element in tempSum which will do XOR of elements
2. Don't take the current element in tempSum
```

And like this we will make 2 recursive calls.
Also we will increase our count by `1` if tempSum becomes equal to `K` because that's what problem says.

we can do this by

```
if(idx == N){
        return tempSum == K;
}
```

So if tempSum is equal to K then it will return `1` otherwise `0`.

also our recursive calls will look like this:

```
int include = solve(arr,N,K,dp,idx+1,tempSum^arr[idx]);
int exclude = solve(arr,N,K,dp,idx+1,tempSum);
```

At last we will return sum of both because we need all subsets

```
return include+exclude;
```

optimising further

⟶ we can store the answers in the dp array using memoization and then use them if we find any other same function call for which we have found answer already.

So code will look like this:

```
if(dp[idx][tempSum] != -1){ // Means there is a already computed answer in any previous same function call
        return dp[idx][tempSum];
}
// Find include and exclude variables from above logic
return dp[idx][tempSum] = include+exclude; // store the current answer in dp array so that other function calls can use it.
```

So final code will look like this:

```
int solve(vector<int>&arr,int N,int K,vector<vector<int>>&dp,int
idx,int tempSum){
        if(idx == N){
        return tempSum == K;
        }
        if(dp[idx][tempSum] != -1){
                return dp[idx][tempSum];
        }
        int include = solve(arr,N,K,dp,idx+1,tempSum^arr[idx]);
        int exclude = solve(arr,N,K,dp,idx+1,tempSum);
        return dp[idx][tempSum] = include+exclude;
}
int subsetXOR(vector<int> arr, int N, int K) {
        vector<vector<int>>dp(N,vector<int>(1000,-1)); // we are
initializing dp array with -1
    return solve(arr,N,K,dp,0,0);
}
```