Codeforces – 1006F - Xor-Paths

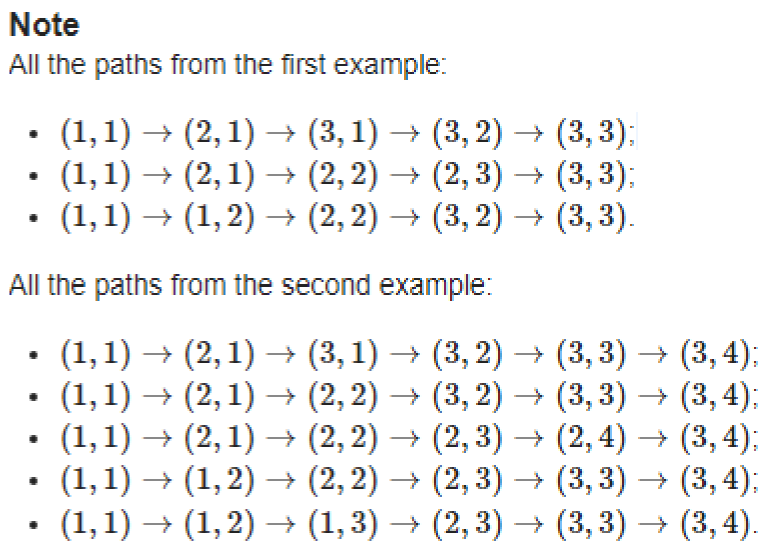
There is a rectangular grid of size n×m. Each cell has a number written on it; the number on the cell (i,j) is a_{ij}. Your task is to calculate the number of paths from the upper-left cell (1,1) to the bottom-right cell (n,m) meeting the following constraints:

- You can move to the right or to the bottom only. Formally, from the cell (i,j) you may move to the cell (i,j+1) or to the cell (i+1,j). The target cell can't be outside of the grid.
- The *xor* of all the numbers on the path from the cell (1,1) to the cell (n,m) must be equal to k.

Find the number of such paths in the given grid.
The first line of the input contains three integers n, mm and k ($1 \le n, m \le 20$, $0 \le k \le 10^{18}$) — the height and the width of the grid, and the number k.
The next n lines contain m integers each, the j-th element in the i-th line is $a_{i,j}$ ($0 \le a_{i,j} \le 10^{18}$).
Print one integer — the number of paths from (1,1) to (n,m) with *xor* sum equal to k.

Input 1:
3 3 11
2 1 5
7 10 0
12 6 4
Output 1: 3

Input 2:
3 4 2
1 3 3 3
0 3 3 2
3 0 1 1
Output 2: 5


Note

All the paths from the first example:

- $(1,1) \to (2,1) \to (3,1) \to (3,2) \to (3,3)$;
- $(1,1) \to (2,1) \to (2,2) \to (2,3) \to (3,3)$;
- $(1,1) \to (1,2) \to (2,2) \to (3,2) \to (3,3)$.

All the paths from the second example:

- $(1,1) \to (2,1) \to (3,1) \to (3,2) \to (3,3) \to (3,4)$;
- $(1,1) \to (2,1) \to (2,2) \to (3,2) \to (3,3) \to (3,4)$;
- $(1,1) \to (2,1) \to (2,2) \to (2,3) \to (2,4) \to (3,4)$;
- $(1,1) \to (1,2) \to (2,2) \to (2,3) \to (3,3) \to (3,4)$;
- $(1,1) \to (1,2) \to (1,3) \to (2,3) \to (3,3) \to (3,4)$.

Idea:

This is a typical problem on the meet-in-the-middle technique. The number of moves we will made equals n+m−2. So if n+m would be small enough (25 is the upper bound, I think), then we can just run recursive backtracking in $O(2^{n+m-2})$ or in $O(({}^{n+m-2}C_{m-1}).(n+m-2))$ to iterate over all binary masks of lengths (n+m−2) containing exactly m−1 ones and check each path described by such mask (0 in this mask is the move to the bottom and 1 is the move to the right) if its xor is k.

But it is too slow. So let's split this mask of n+m−2 bits into two parts — the left part will consist of mid=$\lfloor(n+m-2) / 2\rfloor$ bits and the right part will consist of (n+m−2−mid) bits. Note that each left mask (and each right mask too) uniquely describes the endpoint of the path and the path itself.

Let's carry n×m associative arrays cnt where $cnt_{x,y,c}$ for the endpoint (x,y) and xor c will denote the number of paths which end in the cell (x,y) having xor c.

Let's run recursive backtracking which will iterate over paths starting from the cell (1,1) and move to the right or to the bottom and maintain xor of the path. If we made mid moves and we are currently in the cell (x,y) with xor c right now, set $cnt_{x,y,c}$:= $cnt_{x,y,c}$ +1 and return from the function. Otherwise try to move to the bottom or to the right changing xor as needed.

Let's run another recursive backtracking which will iterate over paths starting from the cell (n,m) and move to the left or to the top and maintain xor of the path except the last cell. The same, if we made (n+m−2−mid) moves and we are currently in the cell (x,y) with xor c right now, let's add $cnt_{x,y,k}$ to the answer (obvious, that way we "complement" our xor from the right part of the path with the suitable xor from the left part of the path). Otherwise try to move to the left or to the top changing xor as needed.

So, this is the meet-in-the-middle technique (at least the way I code it).

Overall complexity is $O(2^{(n+m-2)/2} \cdot (n+m-2) / 2)$.

Sample Code:

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
int n,m,middle;
ll k,a[21][21],ans=0;
unordered_map<ll,ll>mp[21][21];
void dfs1(int r,int c,ll v){
    if(r>n||c>m)return;
    if(r+c==middle){ mp[r][c][v^a[r][c]]++; return; }
    dfs1(r+1,c,v^a[r][c]);
    dfs1(r,c+1,v^a[r][c]);
}
void dfs2(int r,int c,ll v){
    if(r<0||c<0)return;
    if(r+c==middle){ ans += mp[r][c][v^k]; return; }
    dfs2(r-1,c,v^a[r][c]);
    dfs2(r,c-1,v^a[r][c]);
}
int main(){
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    cin>>n>>m>>k;
    for(int i=1; i<=n; i++)
        for(int j=1; j<=m; j++)
            cin>>a[i][j];

    middle = ((n-1+m-1)/2)+2;
    dfs1(1,1,0LL);
    dfs2(n,m,0LL);
    cout<<ans<<endl;
    return 0;
}
```