```
# Maximum square in a given grid
if(len(matrix) == 0):
    return 0
# result
maxSquare = 0
# Init and Base Case setting
DP = [[0 for x in range(0, len(matrix[0]))] for y in range(0,
len(matrix))]
for y in range(0, len(matrix)):
    for x in range(0, len(matrix[y])):
        if(y == 0 or x == 0):
            DP[y][x] = int(matrix[y][x])
            if(maxSquare < DP[y][x]):
                maxSquare = DP[y][x]
# DP[y][x] = min(matrix[y-1][x-1], matrix[y][x-1], matrix[y-1][x]) if
matrix[y][x] == 1
for y in range(1, len(matrix)):
    for x in range(1, len(matrix[y])):
        if(matrix[y][x] == '1'):
            DP[y][x] = int(min(min(DP[y-1][x], DP[y][x-1]), DP[y-1][x-1]))
+ 1
            if(maxSquare < DP[y][x]):
                maxSquare = DP[y][x]
return maxSquare**2
```

```python
## Paths to a goal

# Recursive
string = input()
limit = int(input())
start = int(input())
end = int(input())

seqs = set()
ans = [0]
def solve(curr, target, pointer, string, ans, path):
    if(curr == target):
        if(path not in seqs):
            ans[0] += 1
            seqs.add(path)
    if(pointer >= len(string)):
        return
    if(string[pointer] == 'l'):
        if(curr - 1 >= 0):
            solve(curr - 1, target, pointer + 1, string, ans, path + 'l')
    elif(string[pointer] == 'r'):
        if(curr < limit):
            solve(curr + 1, target, pointer + 1, string, ans, path + 'r')
    solve(curr, target, pointer + 1, string, ans, path)

solve(start, end, 0, string, ans, '')
print(ans[0] % (10**9 + 7))
```

```
# DP -- not working properly
string = input()
limit = int(input())
start = int(input())
end = int(input())

def solve(string, index, Flag, start, end, limit, DP):
    # Base Case
    if(start > limit or start < 0):
        return 0
    # Memoization
    if(DP[index][start][Flag] != -1):
        return DP[index][start][Flag]

    length = len(string)
    if(index >= length):
        return 0
    res = 0
    if((string[index] == 'l' and Flag == 0) or (string[index] == 'r' and
Flag != 0)):
        temp = 1
        if(string[index] == 'l'):
            temp -= 1
        res = solve(string, index + 1, Flag, start + temp, end, limit, DP)
+ solve(string, index + 1, 1 - Flag, start + temp, end, limit, DP)
        if(start + temp == end):
            res += 1
    else:
        res = solve(string, index + 1, Flag, start, end, limit, DP)

    DP[index][start][Flag] = res
    return res

DP = [[[-1, -1] for x in range(0, 100)] for y in range(0, 101)]
res = solve(string, 0, 0, start, end, limit, DP) + solve(string, 0, 1,
start, end, limit, DP)
print(res)
```

```java
## JAVA DP -- works properly
/*package whatever //do not write package name here */

import java.io.*;

class GFG {

    public static void main(String[] args) {

        //Scanner sc = new Scanner(System.in);
        String str = "rrlrlr";
        int n = 6;
        int s = 1;
        int d = 3;
        // Assuming n and str.len() < 100
        int[][][] dp = new int[101][100][2];
        for (int i = 0; i < 100; i++)
            for (int j = 0; j < 100; j++)
                for (int z = 0; z < 2; z++)
                    dp[i][j][z] = -1;

        int res = solve(str, 0 , 0, s, d, n, dp) + solve(str, 0, 1, s, d,
n, dp);
        System.out.println(res);

    }
    static int solve(String str, int idx, int flag, int s, int d, int n,
int[][][] dp){
        if(s > n || s < 0)
            return 0;
        if(dp[idx][s][flag] != -1) {
            return dp[idx][s][flag];
        }
        int len = str.length();
        if(idx >= len)
            return 0;
        int res;
        if( (str.charAt(idx) == 'l' && flag == 0 ) || ( str.charAt(idx) ==
'r' && flag != 0) ){
            int f = 1;
```

```
            if(str.charAt(idx) == '1')
                f = -1;
            res = solve(str, idx + 1, flag, s + f, d, n, dp) + solve(str,
idx + 1, 1 - flag, s + f, d, n, dp);
            // Do a modulo here if needed
            if(s + f == d)
                res++;
        }
        else{
            res = solve(str, idx + 1, flag, s, d, n, dp);
        }
        dp[idx][s][flag] = res;
        return res;
    }
}
## Reconstruct Arrays given totalCost and n,m
Ns = list(map(int, input().split()))
Ms = list(map(int, input().split()))
Costs = list(map(int, input().split()))
def solve(n, m, totalCost):
    # Cumm[i][j][k] = DP[i][1][k] + DP[i][2][k] .. jth
    DP = [[[0 for _ in range(0, totalCost + 1)] for x in range(0, m + 1)]
for y in range(0, n + 1)]
    Cumm = [[[0 for _ in range(0, totalCost + 1)] for x in range(0, m +
1)] for y in range(0, n + 1)]

    for x in range(1, m + 1):
        DP[1][x][0] = 1
        Cumm[1][x][0] = x

    for y in range(2, n + 1):
        for x in range(1, m + 1):
            for w in range(0, totalCost + 1):
                DP[y][x][w] = (x * DP[y - 1][x][w]) % (10**9 + 7)
                DP[y][x][w] = (DP[y][x][w]+Cumm[y-1][x-1][w-1])%(10**9+7)
                Cumm[y][x][w] = (DP[y][x][w] + Cumm[y][x-1][w])%(10**9+ 7)
    print(Cumm[n][m][totalCost])
for i in range(0, len(Ns)):
    solve(Ns[i], Ms[i], Costs[i])
```

```python
## XOR Consecutive pairs subsequence
N = int(input())
Arr = list(map(int, input().strip().split()))
K = int(input())

def solve(Arr):
    from collections import defaultdict
    Dict = defaultdict(lambda : 0)
    maxLen = 1
    for i in range(0, len(Arr)):
        tempLen = Dict[Arr[i] ^ K] + 1
        Dict[Arr[i]] = tempLen
        maxLen = max(maxLen, tempLen)
    return maxLen


print(solve(Arr))

# Fenwick Tree - Insert elements with least cost
class BIT:

    def __init__(self, size):
        self.size = size
        self.Tree = [0 for x in range(0, self.size)]

    def update(self, index, value):
        while(index < self.size):
            self.Tree[index] += value
            index += index & (-index)

    def sum(self, index):
        total = 0
        while(index > 0):
            total += self.Tree[index]
            index -= index & (-index)
        return total
```

```
def solve(Arr):
    bitTree = BIT(10**2)
    res = 0
    for i in range(0, len(Arr)):
        bitTree.update(Arr[i], 1)
        larger = i + 1 - bitTree.sum(Arr[i])
        smaller = bitTree.sum(Arr[i] - 1)
        res += 2 * min(larger, smaller) + 1
    return res

Arr = list(map(int, input().strip().split()))
print(solve(Arr))
```