Paint the Wall

Abhishek has moved to a new apartment and is happy. But the wall of his room looks white and dull. So, he decides to paint the wall. The wall of the room can be described by two integers **N** and **M** denoting height and width respectively. Abhishek is creative and doesn't want to color the whole wall with one color.

So Abhishek came up with an idea of coloring the wall. He says let's consider the wall to be a matrix of **N x M** blocks. He will paint $\mathbf{R_i}$ blocks in each row ($0 \leq i < N$), also he wants only $\mathbf{C_i}$ blocks to be painted in each column ($0 \leq i < M$). He will keep the rest of the blocks uncolored. Now tell Abhishek, In how many ways can he paint the wall.

Input:-

First line of the input contains an integer **T** denoting the number of test cases. **T** test cases follow. Each test case contains three lines.

First line contains two space separated integers **N** and **M** respectively.

Second line contains **N** space separated integers denoting number of blocks to be paint in each row.

Third line contains **M** space separated integers denoting number of blocks to be painted in each column.

Output:-

For each test case,

If the number of ways to paint the wall is greater than one, then print the number of ways to paint the wall.

If number of ways to paint the wall is one, then print the way the wall can be painted using a matrix of **N x M** size. Print **‘0’** for uncoloured block and **‘1’** for the block to be painted.

If number of ways to paint the wall is zero, the print the string **“No Possible Way”**.

Constraints:-

- $1 \leq T \leq 10$
- $1 \leq N, M \leq 8$
- $0 \leq R_i \leq M$
- $0 \leq C_i \leq N$

Example

Input:
3
4 4
1 1 1 1
1 1 1 1
3 3
3 2 3
3 3 2
4 4
1 2 1 3
3 2 2 1

Output:
24
1 1 1
1 1 0
1 1 1
No Possible Way

Explanation

FIrst test case has a 4 x 4 wall. You want to color 1 block in each row and only 1 block in each column. There are 24 valid ways to do this. Hence 24 is the output.

Second test case has a 3 x 3 wall. And there is only one way to paint that wall. Hence we output a matrix depicting how the wall should be colored.

Third test case has a 4 x 4 wall. And there is no possible way to paint the wall satisfying all the given conditions. Hence we output the string “No Possible Way”.

Editorial

PROBLEM LINK:

Contest

Practice

Author: **Abhishek Goyal**

Editorialist: **Abhishek Goyal**

DIFFICULTY:

Hard.

PREREQUISITES:

Dynamic programming, Backtracking.

PROBLEM:

Given the number of blocks to be coloured in each row and column. Find the number of ways to color the wall.

Basic Naive Solution :

A very basic solution to the problem is to bruteforce and count all the valid combinations. But the efficiency of this solution will be very bad. There are total **N x M** blocks. Each of which can be either a **0** or a **1**. This makes the time complexity of this solution to be **O($2^{N \times M}$)**. This solution will roughly work till **N = 5** with only one test case. Here is this solution in **C++**.

Code :

```
#include <bits/stdc++.h>
using namespace std;

int sum(vector<int> v) {
   int sum = 0;
   for(auto i : v)
       sum += i;
   return sum;
}
void print_wall(vector<string> wall) {
   for(auto i : wall) {
       for(auto j : i)
           cout << j << " ";
       cout << endl;
   }
}
void paint(vector<string> &wall, vector<int> &row, vector<int> &col, int
&cnt, vector<string> &fvp, int block = 0) {
   int n = wall.size(), m = wall[0].size();
   int i = block / m, j = block % m;
   if(sum(row) + sum(col) == 0) {
       ++cnt;
       if(cnt == 1)
           fvp = wall;
       return;
   } if(block == n * m) return;
   paint(wall, row, col, cnt, fvp, block + 1);
   if(row[i] > 0 && col[j] > 0) {
       wall[i][j] = '1';
       row[i]--;
       col[j]--;
       paint(wall, row, col, cnt, fvp, block + 1);
```

```
            wall[i][j] = '0';
            row[i]++;
            col[j]++;
        }
    }
    int main() {
        int n, m, t;
        cin >> t;
        while(t--) {
            cin >> n >> m;
            vector<int> row(n), col(m);
            for(int i = 0; i < n; i++)
                cin >> row[i];
            for(int i = 0; i < m; i++)
                cin >> col[i];
            vector<string> wall(n, string(m, '0'));
            vector<string> first_valid_paint;
            int cnt_valid_paint = 0;
            paint(wall, row, col, cnt_valid_paint, first_valid_paint);
            if(cnt_valid_paint == 1)
                print_wall(first_valid_paint);
            else if(cnt_valid_paint == 0)
                cout << "No Possible Way\n";
            else
                cout << cnt_valid_paint << "\n";
        }
    }
```

A Better Solution using Dynamic Programing :

A better approach to the problem will be to solve it rowwise instead of solving it blockwise. We can do the work for a row and use the function recursively to do the work for other rows instead of doing it for just a block and calling the function to do it for all other blocks.

To implement this, we can first create a 2D DP array. **DP[x][y]** of this array stores a list of Binary Strings that have **y / x** bits set. For example, If **x = 3** & **y = 2** the DP Array will contain the **List - {“011”, “101”, “110”}** that is a list of all possible 3 sized binary strings that have 2 bits set. Creating this 2D array is very simple. First we can fill base case that is when **y = 0** , the number of set bits in a **x** bit number is **0**. So **DP[x][0] = {“00...00”}**, that is a **x** length string with all **‘0’**s. We can find all combinations of length **x** with **y** bits set by prefixing **‘0’** to all combinations of length **x - 1** with **y** bits set and **‘1’** to all combinations of length **x - 1** with **n - 1** bits set.

After creating the DP array we can use backtracking to construct the solution rowwise. You must be thinking that how does this DP array help us in backtracking. But as you know we have a matrix of **N x M** size. So we have **N** rows of **M** size and we also know how much blocks are to be painted in each row. We can use this DP array to get **DP[M][R_i]** that is list of binary strings of size **M** which will have R_i bits set, in other words we have a list of all possible states of row of **M** size with R_i blocks painted. We will use them one by one and call the function for the next row to be filled while checking that we do not exceed the number of blocks to be painted in each column. When the current row that we would be working on equals **N**, we will know that we have arrived at a valid solution. Here is this solution implemented in **C++**.

Code :

```
#include <bits/stdc++.h>
using namespace std;

vector<string> DP[16][16];
void nSetBits() {
   int k = 15;
   string str = "";
   for (int i = 0; i <= k; i++) {
       DP[i][0].push_back(str);
       str = str + char(0);
   }
   // DP[k][n] will store all k-bit numbers
   // with n-bits set
   for (int i = 1; i <= k; i++) {
       for (int n = 1; n <= i; n++) {
           for (string str : DP[i - 1][n])
               DP[i][n].push_back(char(0) + str);
           for (string str : DP[i - 1][n - 1])
               DP[i][n].push_back(char(1) + str);
       }
   }
}
int sum(vector<int> v) {
   int sum = 0;
   for(auto i : v)
       sum += i;
   return sum;
}
void print_wall(vector<string> wall) {
   for(auto i : wall) {
       for(int j : i)
           cout << j << " ";
       cout << "\n";
   }
}
```

```
void paint(vector<string> &wall, vector<int> &row, vector<int> &col, int
&cnt, vector<string> &fvp, int i = 0) {
   int n = row.size(), m = col.size();
   if(i == n) {
       if(sum(row) + sum(col) == 0) {
           ++cnt;
           if(cnt == 1)
               fvp = wall;
       } return;
   } int row_i = row[i];
   for(auto s : DP[m][row_i]) {
       row[i] = 0;
       wall[i] = s;
       vector<int> init_col(col);
       bool s_valid = true;
       for(int i = 0; i < col.size(); i++) {
           if(s[i]) {
               if(col[i] > 0)
                   col[i]--;
               else {
                   s_valid = false;
                   break;
               }
           }
       }
       if(s_valid)
           paint(wall, row, col, cnt, fvp, i + 1);
       col = init_col;
       row[i] = row_i;
   }
}
int main() {
   nSetBits();
   int n, m, t;
   cin >> t;
   while(t--) {
       cin >> n >> m;
```

```
        vector<int> row(n), col(m);
        for(int i = 0; i < n; i++)
            cin >> row[i];
        for(int i = 0; i < m; i++)
            cin >> col[i];
        vector<string> wall(n, string(m, 0));
        vector<string> first_valid_paint;
        int cnt_valid_paint = 0;
        paint(wall, row, col, cnt_valid_paint, first_valid_paint);
        if(cnt_valid_paint == 1)
            print_wall(first_valid_paint);
        else if(cnt_valid_paint == 0)
            cout << "No Possible Way\n";
        else
            cout << cnt_valid_paint << "\n";
    }
}
```