```
474 Ones and Zeroes
```

In the computer world, use restricted resource you have to generate maximum benefit is what we always want to pursue.

For now, suppose you are a dominator of **m** `0s` and **n** `1s` respectively. On the other hand, there is an array with strings consisting of only `0s` and `1s`.

Now your task is to find the maximum number of strings that you can form with given **m** `0s` and **n** `1s`. Each `0` and `1` can be used at most **once**.

Note:

1. The given numbers of `0s` and `1s` will both not exceed `100`
2. The size of given string array won't exceed `600`.

Example 1:

```
Input: Array = {"10", "0001", "111001", "1", "0"}, m = 5, n = 3
Output: 4

Explanation: This are totally 4 strings can be formed by the using of 5 0s and 3 1s, which are
“10,”0001”,”1”,”0”
```

Example 2:

```
Input: Array = {"10", "0", "1"}, m = 1, n = 1
Output: 2

Explanation: You could form "10", but then you'd have nothing left. Better form "0" and "1".
```

Intution :
This problem wants us to **select maximum** no of strings **given some number of ones and zeroes** .
So this is a typical 0/1 Knapsack problem
But the twist is that it has 2 constraints

---

approach :
for every string :
 starting from 0 number of **ones** and 0 number of **zeroes**
 To m number of **zeroes** and n number **ones**
 find the number of strings possible with this much resource

using recursive formula :
dp[i][j][k] = max(dp[i-1][j][k], 1 + dp[i-1][j-z][k-o])
//If you pay attention the recursive formula is exaclty same as 0/1 knapsack except one more additional constraint
where :
i <---- tracks string
j <---- tracks number of zeores for i th string
k <--- tracks number of ones for j th string
z <--- number of zeores in string i
o <---- number of ones in string i
if(o > j || z > k) then dp[i][j][k] = dp[i-1][j][k]   // implying resources are not enough

---

CODE :

```
class Solution {
public:
    int findMaxForm(vector<string>& strs, int m, int n) {
        if(strs.size() == 0) return 0;
        int dp[strs.size()+1][m+1][n+1];
        for(int s = 0;s <= strs.size();s++)
            for(int i=0;i<=m;i++)
                for(int j=0;j<=n;j++){
                    if(s == 0 || (j==0 && i==0)) dp[s][i][j] = 0;
                    else{
                    int ct0 = 0, ct1 = 0;
                    for(char a : strs[s-1])
                    {
                        if(a == '0')
                            ct0++;
                        else
                            ct1++;
                    }
                    if(ct0 > i || ct1 > j) dp[s][i][j] = dp[s-1][i][j];
                    else
                        dp[s][i][j] = max(1 + dp[s-1][i-ct0][j-ct1],dp[s-1][i][j]);
                    }
                }
        return dp[strs.size()][m][n];
    }
};
```