在计算机界中，我们总是追求用有限的资源获取最大的收益。

现在，假设你分别支配着 **m** 个 `0` 和 **n** 个 `1`。另外，还有一个仅包含 `0` 和 `1` 字符串的数组。

你的任务是使用给定的 **m** 个 `0` 和 **n** 个 `1` ，找到能拼出存在于数组中的字符串的最大数量。每个 `0` 和 `1` 至多被使用**一次**。

注意:

1. 给定 `0` 和 `1` 的数量都不会超过 `100`。
2. 给定字符串数组的长度不会超过 `600`。

示例 1:

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

解释： 总共 `4` 个字符串可以通过 `5` 个 `0` 和 `3` 个 `1` 拼出，即 `"10","0001","1","0"` 。

示例 2:

输入： `Array = {"10", "0", "1"}, m = 1, n = 1`
输出： `2`

解释： 你可以拼出 `"10"`，但之后就没有剩余数字了。更好的选择是拼出 `"0"` 和 `"1"` 。

这道题是一道典型的应用DP来解的题，如果我们看到这种求总数，而不是列出所有情况的题，十有八九都是用DP来解，重中之重就是在于找出递推式。如果你第一反应没有想到用DP来做，想得是用贪心算法来做，比如先给字符串数组排个序，让长度小的字符串在前面，然后遍历每个字符串，遇到0或者1就将对应的m和n的值减小，这种方法在有的时候是不对的，比如对于{"11", "01", "10"}，m=2，n=2这个例子，我们将遍历完“11”的时候，把1用完了，那么对于后面两个字符串就没法处理了，而其实正确的答案是应该组成后面两个字符串才对。所以我们需要建立一个二维的DP数组，其中dp[i][j]表示有i个0和j个1时能组成的最多字符串的个数，而对于当前遍历到的字符串，我们统计出其中0和1的个数为zeros和ones，然后dp[i - zeros][j - ones]表示当前的i和j减去zeros和ones之前能拼成字符串的个数，那么加上当前的zeros和ones就是当前dp[i][j]可以达到的个数，我们跟其原有数值对比取较大值即可，所以递推式如下：

dp[i][j] = max(dp[i][j], dp[i - zeros][j - ones] + 1);

有了递推式，我们就可以很容易的写出代码如下：

```
class Solution {
public:
    int findMaxForm(vector<string>& strs, int m, int n) {
        vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
        for (string str : strs) {
            int zeros = 0, ones = 0;
            for (char c : str) (c == '0') ? ++zeros : ++ones;
            for (int i = m; i >= zeros; --i) {
```

```
                for (int j = n; j >= ones; --j) {
                    dp[i][j] = max(dp[i][j], dp[i - zeros][j - ones] +
1);
                }
            }
        }
        return dp[m][n];
    }
};
```