

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".

Solution 1

Time Complexity: $O(kl + kmn)$, where k is the length of input string array and l is the average length of a string within the array.

```
public int findMaxForm(String[] strs, int m, int n) {
    int[][] dp = new int[m+1][n+1];
    for (String s : strs) {
        int[] count = count(s);
        for (int i=m; i>=count[0]; i--)
            for (int j=n; j>=count[1]; j--)
                dp[i][j] = Math.max(1 + dp[i-count[0]][j-count[1]], dp[i][j]);
    }
    return dp[m][n];
}

public int[] count(String str) {
    int[] res = new int[2];
    for (int i=0; i<str.length(); i++)
        res[str.charAt(i) - '0']++;
    return res;
}
```

Thanks [@shawngao](#) for some ways to make this solution more concise.

written by [compton_scatter](#) original link [here](#)

Solution 2

```
int findMaxForm(vector<string>& strs, int m, int n) {
    vector<vector<int>> memo(m+1, vector<int>(n+1, 0));
    int numZeroes, numOnes;

    for (auto &s : strs) {
        numZeroes = numOnes = 0;
        // count number of zeroes and ones in current string
        for (auto c : s) {
            if (c == '0')
                numZeroes++;
            else if (c == '1')
                numOnes++;
        }

        // memo[i][j] = the max number of strings that can be formed with i 0's and j
        // 1's
        // from the first few strings up to the current string s
        // Catch: have to go from bottom right to top left
        // Why? If a cell in the memo is updated(because s is selected),
        // we should be adding 1 to memo[i][j] from the previous iteration (when we
        // are not considering s)
        // If we go from top left to bottom right, we would be using results from this
        // iteration => overcounting
        for (int i = m; i >= numZeroes; i--) {
            for (int j = n; j >= numOnes; j--) {
                memo[i][j] = max(memo[i][j], memo[i - numZeroes][j - numOnes] + 1);
            }
        }
    }
    return memo[m][n];
}
```

written by [yangluphil](#) original link [here](#)

Solution 3

[UPDATE]:

This is not a working solution as pointed out by [@andhddn](#) below.

I have a greedy algorithm that is accepted. Can't prove the correctness though. Main idea is to use the shortest string first. We sort all the string based on the number of 1s it has once, and then based on the 0s second time. Each time we try to consume as much string as possible. Result is the bigger of the 2 iterations.

Any counter example/ proof of correctness is appreciated

```
public class Solution {
    public int findMaxForm(String[] strs, int m, int n) {
        int result = 0;
        if(strs.length == 0) return 0;
        PriorityQueue<Element> oneQueue = new PriorityQueue<Element>(new oneComparator());
        PriorityQueue<Element> zeroQueue = new PriorityQueue<Element>(new zeroComparator());

        for(String s : strs){
            int ones = 0;
            int zeros = 0;
            for(char c : s.toCharArray()){
                if(c == '0') {
                    zeros++;
                }else{
                    ones++;
                }
            }
            Element e = new Element(ones, zeros);
            oneQueue.offer(e);
            zeroQueue.offer(e);
        }
        int numZero = m;
        int numOne = n;
        /*sort the string based on # of 0 each string has and try to consume as m any string as possible*/
        while(!zeroQueue.isEmpty()){
            if(numZero >= zeroQueue.peek().zero && numOne >= zeroQueue.peek().one){
                result++;
                numZero -= zeroQueue.peek().zero;
                numOne -= zeroQueue.peek().one;
            }

            zeroQueue.poll();
        }
        int secondResult = 0;
        numZero = m;
        numOne = n;
        /*sort the string based on # of 1 each string has and try to consume as m any string as possible*/
        while(!oneQueue.isEmpty()){
            if(numOne >= oneQueue.peek().one && numZero >= oneQueue.peek().zero){
```

} ' ' '

written by [DonaldTrump](#) original link [here](#)

From **LeetCoder**.