动态规划 04

474.一和零

题目

力扣题目链接

中等 20min

给你一个二进制字符串数组 `strs` 和两个整数 `m` 和 `n` 。

请你找出并返回 `strs` 的最大子集的长度，该子集中 **最多** 有 `m` 个 `0` 和 `n` 个 `1` 。

如果 `x` 的所有元素也是 `y` 的元素，集合 `x` 是集合 `y` 的 **子集** 。

示例 1：

```
输入：strs = ["10", "0001", "111001", "1", "0"], m = 5, n = 3
输出：4
解释：最多有 5 个 0 和 3 个 1 的最大子集是 {"10","0001","1","0"} ，因此答案是 4 。
其他满足题意但较小的子集包括 {"0001","1"} 和 {"10","1","0"} 。{"111001"} 不满足题意，因为它含 4 个 1 ，大于 n 的值 3 。
```

示例 2：

```
输入：strs = ["10", "0", "1"], m = 1, n = 1
输出：2
解释：最大的子集是 {"0", "1"} ，所以答案是 2 。
```

提示：

- `1 <= strs.length <= 600`
- `1 <= strs[i].length <= 100`
- `strs[i]` 仅由 `'0'` 和 `'1'` 组成
- `1 <= m, n <= 100`

题解

这题乍一看有两个数据，但是这俩玩意必须一起选，所以其实是有两个维度的01背包问题！

一个是m 一个是n，而不同长度的字符串就是不同大小的待装物品。

1.确定dp数组:$dp[i][j]$：最多有i个0和j个1的`strs`的最大子集的大小为$dp[i][j]$。

2.确定递推公式：

$dp[i][j]$可以由前一个`strs`里的字符串推导出来

计算出当前的字符串有`zeroNum`个0，`oneNum`个1

$dp[i][j]$就可以是$dp[i-zeroNum][j-oneNum]+1$

然后我们在遍历的过程中，将这个值和不考虑这个物品时的大小比较，取$dp[i][j]$的最大值。

所以递推公式：$dp[i][j] = max(dp[i][j], dp[i - zeroNum][j - oneNum] + 1);$

此时大家可以回想一下01背包的递推公式：$dp[j] = max(dp[j], dp[j - weight[i]] + value[i]);$

> 对比一下就会发现，字符串的`zeroNum`和`oneNum`相当于物品的重量（$weight[i]$），字符串本身的个数相当于物品的价值（$value[i]$）。

这就是一个典型的01背包！ 只不过物品的重量有了两个维度而已，相当于三维的01背包，压缩之后是二维的。

3. dp数组如何初始化：01背包的dp数组初始化为0就可以。

4. 确定遍历顺序：01背包外层for循环遍历物品，内层for循环遍历背包容量且从后向前遍历（因为压缩了数组）

   那么本题也是，物品就是`strs`里的字符串，背包容量就是题目描述中的`m`和`n`。

```
class Solution {
public:
    int findMaxForm(vector<string>& strs, int m, int n) {
        vector<vector<int>> dp(m + 1, vector<int> (n + 1, 0)); // 默认初始化0
        for (string str : strs) { // 遍历物品
            int oneNum = 0, zeroNum = 0;
            for (char c : str) {
                if (c == '0') zeroNum++;
                else oneNum++;
            }
            for (int i = m; i >= zeroNum; i--) { // 遍历背包容量且从后向前遍历!
                for (int j = n; j >= oneNum; j--) {
                    dp[i][j] = max(dp[i][j], dp[i - zeroNum][j - oneNum] + 1);
                }
            }
        }
        return dp[m][n];
    }
};
```