如果 map 大小为 $M \times N$，经典动态规划方法的时间复杂度为 $O(M \times N)$，额外空间复杂度为 $O(M \times N)$。结合空间压缩之后可以将额外空间复杂度降至 $O(\min\{M,N\})$。空间压缩的原理请读者参考本书“矩阵的最小路径和”问题，这里不再详述。请参看如下代码中的 minHP2 方法。

```
public static int minHP2(int[][] m) {
        if (m == null || m.length == 0 || m[0] == null || m[0].length == 0) {
                return 1;
        }
        int more = Math.max(m.length, m[0].length);
        int less = Math.min(m.length, m[0].length);
        boolean rowmore = more == m.length;
        int[] dp = new int[less];
        int tmp = m[m.length - 1][m[0].length - 1];
        dp[less - 1] = tmp > 0 ? 1 : -tmp + 1;
        int row = 0;
        int col = 0;
        for (int j = less - 2; j >= 0; j--) {
                row = rowmore ? more - 1 : j;
                col = rowmore ? j : more - 1;
                dp[j] = Math.max(dp[j + 1] - m[row][col], 1);
        }
        int choosen1 = 0;
        int choosen2 = 0;
        for (int i = more - 2; i >= 0; i--) {
                row = rowmore ? i : less - 1;
                col = rowmore ? less - 1 : i;
                dp[less - 1] = Math.max(dp[less - 1] - m[row][col], 1);
                for (int j = less - 2; j >= 0; j--) {
                        row = rowmore ? i : j;
                        col = rowmore ? j : i;
                        choosen1 = Math.max(dp[j] - m[row][col], 1);
                        choosen2 = Math.max(dp[j + 1] - m[row][col], 1);
                        dp[j] = Math.min(choosen1, choosen2);
                }
        }
        return dp[0];
}
```

数字字符串转换为字母组合的种数

【题目】

给定一个字符串 str，str 全部由数字字符组成，如果 str 中某一个或某相邻两个字符组成的子串值在 1~26 之间，则这个子串可以转换为一个字母。规定"1"转换为"A"，"2"转换

为"B"，"3"转换为"C"..."26"转换为"Z"。写一个函数，求 str 有多少种不同的转换结果，并返回种数。

【举例】

str="1111"。

能转换出的结果有"AAAA"、"LAA"、"ALA"、"AAL"和"LL"，返回 5。

str="01"。

"0"没有对应的字母，而"01"根据规定不可转换，返回 0。

str="10"。

能转换出的结果是"J"，返回 1。

【难度】

尉 ★★☆☆

【解答】

暴力递归的方法。假设 str 的长度为 N，先定义递归函数 $p(i)(0 \leqslant i \leqslant N)$。$p(i)$的含义是 str[0..i-1]已经转换完毕，而 str[i..N-1]还没转换的情况下，最终合法的转换种数有多少并返回。特别指出，$p(N)$表示 str[0..N-1]（也就是 str 的整体）都已经转换完，没有后续的字符了，那么合法的转换种数为 1，即 $p(N)=1$。比如，str="111123"，p(4)表示 str[0..3]（即"1111"）已经转换完毕，具体结果是什么不重要，反正已经转换完毕并且不可变，没转换的部分是 str[4..5]（即"23"），可转换的为"BC"或"W"只有两种，所以 $p(4)=2$。$p(6)$表示 str 整体已经转换完毕，所以 $p(6)=1$。那么 $p(i)$如何计算呢？只有以下 4 种情况。

- 如果 i==N。根据上文对 $p(N)=1$ 的解释，直接返回 1。
- 如果不满足情况 1，又有 str[i]=='0'。str[0..i-1]已经转换完毕，而 str[i..N-1]此时又以'0'开头，str[i..N-1]无论怎样都不可能合法转换，所以直接返回 0。
- 如果不满足情况 1 和情况 2，说明 str[i]属于'1'~'9'，str[i]可以转换为'A'~'I'，那么 $p(i)$的值一定包含 $p(i+1)$的值，即 $p(i)=p(i+1)$。
- 如果不满足情况 1 和情况 2，说明 str[i]属于'1'~'9'，如果又有 str[i..i+1]在"10""~"26"之间，str[i..i+1]可以转换为'J'~'Z'，那么 $p(i)$的值一定也包含 $p(i+2)$的值，即 $p(i)+=p(i+2)$。

具体过程请参看如下代码中的 num1 方法。

```
public int num1(String str) {
        if (str == null || str.equals("")) {
                return 0;
        }
        char[] chs = str.toCharArray();
        return process(chs, 0);
}

public int process(char[] chs, int i) {
        if (i == chs.length) {
                return 1;
        }
        if (chs[i] == '0') {
                return 0;
        }
        int res = process(chs, i + 1);
        if (i + 1 < chs.length && (chs[i] - '0') * 10 + chs[i + 1] - '0' < 27) {
                res += process(chs, i + 2);
        }
        return res;
}
```

以上过程中，$p(i)$最多可能会有两个递归分支 $p(i+1)$和 $p(i+2)$，一共有 N 层递归，所以时间复杂度为 $O(2^N)$，额外空间复杂度就是递归使用的函数栈的大小为 $O(N)$。但是研究一下递归函数 p 就会发现，$p(i)$最多依赖 $p(i+1)$和 $p(i+2)$的值，这是可以从后往前进行顺序计算的，也就是先计算 $p(N)$和 $p(N-1)$，然后根据这两个值计算 $p(N-2)$，再根据 $p(N-1)$和 $p(N-2)$计算 $p(N-3)$，最后根据 $p(1)$和 $p(2)$计算出 $p(0)$即可，类似斐波那契数列的求解过程，只不过斐波那契数列是从前往后计算的，这里是从后往前计算而已。具体过程请参看如下代码中的 num2 方法。

```
public int num2(String str) {
        if (str == null || str.equals("")) {
                return 0;
        }
        char[] chs = str.toCharArray();
        int cur = chs[chs.length - 1] == '0' ? 0 : 1;  cur为当前状态，next为上一个状态
        int next = 1;
        int tmp = 0;
        for (int i = chs.length - 2; i >= 0; i--) {
                if (chs[i] == '0') {
                        next = cur;
                        cur = 0;
                } else {
                        tmp = cur;
                        if ((chs[i] - '0') * 10 + chs[i + 1] - '0' < 27) {
                                cur += next;
                        }
```

```
                    next = tmp;
                }
            }
            return cur;
        }
```

因为是顺序计算，所以 num2 方法的时间复杂度为 $O(N)$，同时只用了 cur、next 和 tmp 进行滚动更新，所以额外空间复杂度为 $O(1)$。但是本题并不能像斐波那契数列问题那样用矩阵乘法的优化方法将时间复杂度优化到 $O(\log N)$，这是因为斐波那契数列是严格的 $f(i)=f(i-1)+f(i-2)$，但是本题并不严格，str[i]的具体情况决定了 $p(i)$是等于 0 还是等于 $p(i+1)$，还是等于 $p(i+1)+p(i+2)$。有状态转移的表达式不可以用矩阵乘法将时间复杂度优化到 $O(\log N)$。但如果 str 只由字符'1'和字符'2'组成，比如"12121121212122"，那么就可以使用矩阵乘法的方法将时间复杂度优化为 $O(\log N)$。因为 str[i]都可以单独转换成字母，str[i..i+1]也都可以一起转换成字母，此时一定有 $p(i)=p(i+1)+p(i+2)$。总之，可以使用矩阵乘法的前提是递归表达式不会发生转移。

表达式得到期望结果的组成种数

【题目】

给定一个只由 0（假）、1(真)、&(逻辑与)、|（逻辑或）和^(异或)五种字符组成的字符串 express，再给定一个布尔值 desired。返回 express 能有多少种组合方式，可以达到 desired 的结果。

【举例】

express="1^0|0|1"，desired=false。

只有 1^((0|0)|1)和 1^(0|(0|1))的组合可以得到 false，返回 2。

express="1"，desired=false。

无组合则可以得到 false，返回 0。

【难度】

校 ★★★☆