```
        char[] shorts = chs1.length < chs2.length ? chs1 : chs2;
        if (chs1.length < chs2.length) { // str2 较长就交换 ic 和 dc 的值
                int tmp = ic;
                ic = dc;
                dc = tmp;
        }
        int[] dp = new int[shorts.length + 1];
        for (int i = 1; i <= shorts.length; i++) {
                dp[i] = ic * i;
        }
        for (int i = 1; i <= longs.length; i++) {
                int pre = dp[0]; // pre 表示左上角的值
                dp[0] = dc * i;
                for (int j = 1; j <= shorts.length; j++) {
                        int tmp = dp[j]; // dp[j]没更新前先保存下来
                        if (longs[i - 1] == shorts[j - 1]) {
                                dp[j] = pre;
                        } else {
                                dp[j] = pre + rc;
                        }
                        dp[j] = Math.min(dp[j], dp[j - 1] + ic);
                        dp[j] = Math.min(dp[j], tmp + dc);
                        pre = tmp; // pre 变成 dp[j]没更新前的值
                }
        }
        return dp[shorts.length];
}
```

字符串的交错组成

【题目】

给定三个字符串 str1、str2 和 aim，如果 aim 包含且仅包含来自 str1 和 str2 的所有字符，而且在 aim 中属于 str1 的字符之间保持原来在 str1 中的顺序，属于 str2 的字符之间保持原来在 str2 中的顺序，那么称 aim 是 str1 和 str2 的交错组成。实现一个函数，判断 aim 是否是 str1 和 str2 交错组成。

【举例】

str1="AB"，str2="12"。那么"AB12"、"A1B2"、"A12B"、"1A2B"和"1AB2"等都是 str1 和 str2 的交错组成。

【难度】

校 ★★★☆

【解答】

如果 str1 的长度为 M，str2 的长度为 N，经典动态规划的方法可以达到时间复杂度为 $O(M\times N)$，额外空间复杂度为 $O(M\times N)$。如果结合空间压缩的技巧，可以把额外空间复杂度减至 $O(\min\{M,N\})$。

先来介绍经典动态规划的方法。首先 aim 如果是 str1 和 str2 的交错组成，aim 的长度一定是 $M+N$，否则直接返回 false。然后生成大小为$(M+1)\times(N+1)$布尔类型的矩阵 dp，dp[i][j]的值代表 aim[0..i+j-1]能否被 str1[0..i-1]和 str2[0..j-1]交错组成。计算 dp 矩阵的时候，是从左到右，再从上到下计算的，dp[M][N]也就是 dp 矩阵中最右下角的值，表示 aim 整体能否被 str1 整体和 str2 整体交错组成，也就是最终结果。下面具体说明 dp 矩阵每个位置的值是如何计算的。

1．dp[0][0]=true。aim 为空串时，当然可以被 str1 为空串和 str2 为空串交错组成。

2．矩阵 dp 第一列即 dp[0..M-1][0]。dp[i][0]表示 aim[0..i-1]能否只被 str1[0..i-1]交错组成。如果 aim[0..i-1]等于 str1[0..i-1]，则令 dp[i][0]=true，否则令 dp[i][0]=false。

3．矩阵 dp 第一行即 dp[0][0..N-1]。dp[0][j]表示 aim[0..j-1]能否只被 str2[0..j-1]交错组成。如果 aim[0..j-1]等于 str1[0..j-1]，则令 dp[i][0]=true，否则令 dp[i][0]=false。

4．对其他位置(i,j)，dp[i][j]的值由下面的情况决定。

- dp[i-1][j]代表 aim[0..i+j-2]能否被 str1[0..i-2]和 str2[0..j-1]交错组成，如果可以，那么如果再有 str1[i-1]等于 aim[i+j-1]，说明 str1[i-1]又可以作为交错组成 aim[0..i+j-1]的最后一个字符。令 dp[i][j]=true。
- dp[i][j-1]代表 aim[0..i+j-2]能否被 str1[0..i-1]和 str2[0..j-2]交错组成，如果可以，那么如果再有 str2[j-1]等于 aim[i+j-1]，说明 str1[j-1]又可以作为交错组成 aim[0..i+j-1]的最后一个字符。令 dp[i][j]=true。
- 如果第 1 种情况和第 2 种情况都不满足，令 dp[i][j]=false。

具体过程请参看如下代码中的 isCross1 方法。

```
public boolean isCross1(String str1, String str2, String aim) {
        if (str1 == null || str2 == null || aim == null) {
                return false;
        }
        char[] ch1 = str1.toCharArray();
```

```
        char[] ch2 = str2.toCharArray();
        char[] chaim = aim.toCharArray();
        if (chaim.length != ch1.length + ch2.length) {
                return false;
        }
        boolean[][] dp = new boolean[ch1.length + 1][ch2.length + 1];
        dp[0][0] = true;
        for (int i = 1; i <= ch1.length; i++) {
                if (ch1[i - 1] != chaim[i - 1]) {
                        break;
                }
                dp[i][0] = true;
        }
        for (int j = 1; j <= ch2.length; j++) {
                if (ch2[j - 1] != chaim[j - 1]) {
                        break;
                }
                dp[0][j] = true;
        }
        for (int i = 1; i <= ch1.length; i++) {
                for (int j = 1; j <= ch2.length; j++) {
                    if ((ch1[i - 1] == chaim[i + j - 1] && dp[i - 1][j])
                        || (ch2[j - 1] == chaim[i + j - 1] && dp[i][j - 1])) {
                            dp[i][j] = true;
                    }
                }
        }
        return dp[ch1.length][ch2.length];
}
```

经典动态规划方法结合空间压缩的方法。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。实际进行空间压缩的时候，比较 str1 和 str2 中哪个长度较小，长度较小的那个作为列对应的字符串，然后生成和较短字符串长度一样的一维数组 dp，滚动更新即可。

具体请参看如下代码中的 isCross2 方法。

```
public boolean isCross2(String str1, String str2, String aim) {
        if (str1 == null || str2 == null || aim == null) {
                return false;
        }
        char[] ch1 = str1.toCharArray();
        char[] ch2 = str2.toCharArray();
        char[] chaim = aim.toCharArray();
        if (chaim.length != ch1.length + ch2.length) {
                return false;
        }
        char[] longs = ch1.length >= ch2.length ? ch1 : ch2;
        char[] shorts = ch1.length < ch2.length ? ch1 : ch2;
        boolean[] dp = new boolean[shorts.length + 1];
```

```
        dp[0] = true;
        for (int i = 1; i <= shorts.length; i++) {
            if (shorts[i - 1] != chaim[i - 1]) {
                break;
            }
            dp[i] = true;
        }
        for (int i = 1; i <= longs.length; i++) {
            dp[0] = dp[0] && longs[i - 1] == chaim[i - 1];
            for (int j = 1; j <= shorts.length; j++) {
                if ((longs[i - 1] == chaim[i + j - 1] && dp[j])
                    || (shorts[j - 1] == chaim[i + j - 1] && dp[j - 1])) {
                    dp[j] = true;
                } else {
                    dp[j] = false;
                }
            }
        }
        return dp[shorts.length];
    }
```

龙与地下城游戏问题

【题目】

给定一个二维数组 map，含义是一张地图，例如，如下矩阵：

-2　-3　3

-5　-10　1

0　30　-5

游戏的规则如下：

- 骑士从左上角出发，每次只能向右或向下走，最后到达右下角见到公主。
- 地图中每个位置的值代表骑士要遭遇的事情。如果是负数，说明此处有怪兽，要让骑士损失血量。如果是非负数，代表此处有血瓶，能让骑士回血。
- 骑士从左上角到右下角的过程中，走到任何一个位置时，血量都不能少于 1。

为了保证骑士能见到公主，初始血量至少是多少？根据 map，返回初始血量。

【难度】

尉 ★★☆☆