$$(C_n, C_{n-1}, C_{n-2}) = (C_{n-1}, C_{n-2}, C_{n-3}) \times \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

把前 5 项 C(1)==1，C(2)==2，C(3)==3，C(4)==4，C(5)==6 代入，求出状态矩阵：

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = \begin{vmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{vmatrix}$$

求矩阵之后，当 n>3 时，原来的公式可化简为：

$$(C_n, C_{n-1}, C_{n-2}) = (C_3, C_2, C_1) \times \begin{vmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{vmatrix}^{n-3} = (3,2,1) \times \begin{vmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{vmatrix}^{n-3}$$

接下来的过程又是利用加速矩阵乘法的方式进行实现，具体请参看如下代码中的c3方法。

```
public int c3(int n) {
        if (n < 1) {
                return 0;
        }
        if (n == 1 || n == 2 || n == 3) {
                return n;
        }
        int[][] base = { { 1, 1, 0 }, { 0, 0, 1 }, { 1, 0, 0 } };
        int[][] res = matrixPower(base, n - 3);
        return 3 * res[0][0] + 2 * res[1][0] + res[2][0];
}
```

如果递归式严格符合 $F(n)=a\times F(n\text{-}1)+b\times F(n\text{-}2)+...+k\times F(n\text{-}i)$，那么它就是一个 i 阶的递推式，必然有与 $i\times i$ 的状态矩阵有关的矩阵乘法的表达。一律可以用加速矩阵乘法的动态规划将时间复杂度降为 $O(\log N)$。

矩阵的最小路径和

【题目】

给定一个矩阵 m，从左上角开始每次只能向右或者向下走，最后到达右下角的位置，路径上所有的数字累加起来就是路径和，返回所有的路径中最小的路径和。

【举例】

如果给定的 m 如下：

1　3　5　9

8　1　3　4

5　0　6　1

8　8　4　0

路径 1，3，1，0，6，1，0 是所有路径中路径和最小的，所以返回 12。

【难度】

尉 ★★☆☆

【解答】

经典动态规划方法。假设矩阵 m 的大小为 $M\times N$，行数为 M，列数为 N。先生成大小和 m 一样的矩阵 dp，dp[i][j]的值表示从左上角（即(0,0)）位置走到(i,j)位置的最小路径和。对 m 的第一行的所有位置来说，即$(0,j)(0\leqslant j<N)$，从(0,0)位置走到$(0,j)$位置只能向右走，所以(0,0)位置到$(0,j)$位置的路径和就是 m[0][0..j]这些值的累加结果。同理，对 m 的第一列的所有位置来说，即$(i,0)(0\leqslant i<M)$，从(0,0)位置走到$(i,0)$位置只能向下走，所以(0,0)位置到$(i,0)$位置的路径和就是 m[0..i][0]这些值的累加结果。以题目中的例子来说，dp 第一行和第一列的值如下：

1　4　9　1　8

9

14

22

除第一行和第一列的其他位置(i,j)外，都有左边位置$(i-1,j)$和上边位置$(i,j-1)$。从(0,0)到(i,j)的路径必然经过位置$(i-1,j)$或位置$(i,j-1)$，所以，dp[i][j]=min{dp[i-1][j],dp[i][j-1]}+m[i][j]，含义是比较从(0,0)位置开始，经过$(i-1,j)$位置最终到达(i,j)的最小路径和经过$(i,j-1)$位置最终到达(i,j)的最小路径之间，哪条路径的路径和更小。那么更小的路径和就是 dp[i][j]的值。以题目的例子来说，最终生成的 dp 矩阵如下：

1　4　9　18

9　5　8　12

14　5　11　12

22　13　15　12

除第一行和第一列之外，每一个位置都考虑从左边到达自己的路径和更小还是从上边达到自己的路径和更小。最右下角的值就是整个问题的答案。具体过程请参看如下代码中

的 minPathSum1 方法。

```
public int minPathSum1(int[][] m) {
        if (m == null || m.length == 0 || m[0] == null || m[0].length == 0) {
                return 0;
        }
        int row = m.length;
        int col = m[0].length;
        int[][] dp = new int[row][col];
        dp[0][0] = m[0][0];
        for (int i = 1; i < row; i++) {
                dp[i][0] = dp[i - 1][0] + m[i][0];
        }
        for (int j = 1; j < col; j++) {
                dp[0][j] = dp[0][j - 1] + m[0][j];
        }
        for (int i = 1; i < row; i++) {
                for (int j = 1; j < col; j++) {
                   dp[i][j] = Math.min(dp[i - 1][j], dp[i][j - 1]) + m[i][j];
                }
        }
        return dp[row - 1][col - 1];
}
```

矩阵中一共有 $M\times N$ 个位置，每个位置都计算一次从(0,0)位置达到自己的最小路径和，计算的时候只是比较上边位置的最小路径和与左边位置的最小路径和哪个更小，所以时间复杂度为 $O(M\times N)$，dp 矩阵的大小为 $M\times N$，所以额外空间复杂度为 $O(M\times N)$。

动态规划经过空间压缩后的方法。这道题的经典动态规划方法在经过空间压缩之后，时间复杂度依然是 $O(M\times N)$，但是额外空间复杂度可以从 $O(M\times N)$减小至 $O(\min\{M,N\})$，也就是不使用大小为 $M\times N$ 的 dp 矩阵，而仅仅使用大小为 $\min\{M,N\}$的 arr 数组。具体过程如下（以题目的例子来举例说明）：

1．生成长度为 4 的数组 arr，初始时 arr=[0,0,0,0]，我们知道从(0,0)位置到达 m 中第一行的每个位置，最小路径和就是从(0,0)位置的值开始依次累加的结果，所以依次把 arr 设置为 arr=[1,4,9,18]，此时 arr[j]的值代表从(0,0)位置达到(0,j)位置的最小路径和。

2．步骤 1 中 arr[j]的值代表从(0,0)位置达到(0,j)位置的最小路径和，在这一步中想把 arr[j]的值更新成从(0,0)位置达到(1,j)位置的最小路径和。首先来看 arr[0]，更新之前 arr[0]的值代表(0,0)位置到达(0,0)位置的最小路径和(dp[0][0])，如果想把 arr[0]更新成从(0,0)位置达到(1,0)位置的最小路径和(dp[1][0])，令 arr[0]=arr[0]+m[1][0]=9 即可。然后来看 arr[1]，更新之前 arr[1]的值代表(0,0)位置到达(0,1)位置的最小路径和(dp[0][1])，更新之后想让 arr[1]代表(0,0)位置到达(1,1)位置的最小路径和(dp[1][1])。根据动态规划的求解过程，到达(1,1)位

置有两种选择，一种是从(1,0)位置到达(1,1)位置(dp[1][0]+m[1][1])，另一种是从(0,1)位置到达(1,1)位置(dp[0][1]+m[1][1])，应该选择路径和最小的那个。此时 arr[0]的值已经更新成 dp[1][0]，arr[1]目前还没有更新，所以，arr[1]还是 dp[0][1]，arr[1]=min{arr[0],arr[1]}+m[1][1]=5。更新之后，arr[1]的值变为 dp[1][1]的值。同理，arr[2]=min{arr[1],arr[2]}+m[1][2]，...最终 arr 可以更新成[9,5,8,12]。

3．重复步骤 2 的更新过程，一直到 arr 彻底变成 dp 矩阵的最后一行。整个过程其实就是不断滚动更新 arr 数组，让 arr 依次变成 dp 矩阵每一行的值，最终变成 dp 矩阵最后一行的值。

本题的例子是矩阵 m 的行数等于列数，如果给定的矩阵列数小于行数（$N<M$），依然可以用上面的方法令 arr 更新成 dp 矩阵每一行的值。但如果给定的矩阵行数小于列数（$M<N$），那么就生成长度为 M 的 arr，然后令 arr 更新成 dp 矩阵每一列的值，从左向右滚动过去。以本例来说，如果按列来更新，arr 首先更新成[1,9,14,22]，然后向右滚动更新成[4,5,5,13]，继续向右滚动更新成[9,8,11,15]，最后是[18,12,12,12]。总之，是根据给定矩阵行和列的大小关系决定滚动的方式，始终生成最小长度(min{M,N})的 arr 数组。具体过程请参看如下代码中的 minPathSum2 方法。

```
public int minPathSum2(int[][] m) {
        if (m == null || m.length == 0 || m[0] == null || m[0].length == 0) {
                return 0;
        }
        int more = Math.max(m.length, m[0].length); // 行数与列数较大的那个为 more
        int less = Math.min(m.length, m[0].length); // 行数与列数较小的那个为 less
        boolean rowmore = more == m.length; // 行数是不是大于等于列数
        int[] arr = new int[less]; // 辅助数组的长度仅为行数与列数中的最小值
        arr[0] = m[0][0];
        for (int i = 1; i < less; i++) {
                arr[i] = arr[i - 1] + (rowmore ? m[0][i] : m[i][0]);
        }
        for (int i = 1; i < more; i++) {
                arr[0] = arr[0] + (rowmore ? m[i][0] : m[0][i]);
                for (int j = 1; j < less; j++) {
                        arr[j] = Math.min(arr[j - 1], arr[j])
                                        + (rowmore ? m[i][j] : m[j][i]);
                }
        }
        return arr[less - 1];
}
```

【扩展】

本题压缩空间的方法几乎可以应用到所有需要二维动态规划表的面试题目中，通过一

个数组滚动更新的方式无疑节省了大量的空间。没有优化之前，取得某个位置动态规划值的过程是在矩阵中进行两次寻址，优化后，这一过程只需要一次寻址，程序的常数时间也得到了一定程度的加速。但是空间压缩的方法是有局限性的，本题如果改成“打印具有最小路径和的路径”，那么就不能使用空间压缩的方法。如果类似本题这种需要二维表的动态规划题目，最终目的是想求最优解的具体路径，往往需要完整的动态规划表，但如果只是想求最优解的值，则可以使用空间压缩的方法。因为空间压缩的方法是滚动更新的，会覆盖之前求解的值，让求解轨迹变得不可回溯。希望读者好好研究这种空间压缩的实现技巧，本书还有许多动态规划题目会涉及空间压缩方法的实现。

换钱的最少货币数

【题目】

给定数组 arr，arr 中所有的值都为正数且不重复。每个值代表一种面值的货币，每种面值的货币可以使用任意张，再给定一个整数 aim 代表要找的钱数，求组成 aim 的最少货币数。

【举例】

arr=[5,2,3]，aim=20。

4 张 5 元可以组成 20 元，其他的找钱方案都要使用更多张的货币，所以返回 4。

arr=[5,2,3]，aim=0。

不用任何货币就可以组成 0 元，返回 0。

arr=[3,5]，aim=2。

根本无法组成 2 元，钱不能找开的情况下默认返回-1。

【补充题目】

给定数组 arr，arr 中所有的值都为正数。每个值仅代表一张钱的面值，再给定一个整数 aim 代表要找的钱数，求组成 aim 的最少货币数。

【举例】

arr=[5,2,3]，aim=20。