给定一个填充非负数的*m* × *n*网格，找到一条从左上到右下的路径，该路径*使*沿路径上所有数字的总和*最小*。

**注意：**您只能在任何时间点向下或向右移动。

**例1：**

[[1,3,1]，

[1,5,1]

[4,2,1]

鉴于上面的网格图，返回7。因为路径1→3→1→1→1使总和最小化。

这道题我们同样采用dp方法来做。

首先，原本的dp数组应该是dp[3][3]，我们把dp数组进行压缩成dp[3]。我们以列为单位，一列一列的算。

我们需要处理的数据分为两类，一是边界值，就是i=0 || j=0的情况；一是非边界值，即i!=0 && j!=0的情况。

当为边界值，我们先处理上边界，也就是i=0的时候。我们在上边界的某点路径长度只能从左边过来，所以在该点的路径长度new dp[j]=左边点的路径长度old dp[j] + 该点的路径长度grid[i][j]。当为左边界的时候，也就是j=0的时候，我们在左边界的某点路径长度只能从上边过来，所以在该点的路径长度dp[j]=上边点的路径长度dp[j-1] + 该点的路径长度grid[i][j]。

当为非边界值的时候，我们在该点就可以从左边或者上边来，也就是说有两种的选择，那么我们该如何选择呢？本题目是最小的路径长度，所以我们需要选择最小的路径。左边点的路径old dp[j],上边点的路径dp[j-1],min=Math.min(old dp[j],dp[j-1]).该点的最小路径长度为本点的路径grid[i][j]+min。

注意，我们这里提到一点就是，java中如何求二维数组的长和宽呢？

Int length = grid.length；//二维数组的长

Int wigth = grid[0].length;//二维数组的宽

本题的时间复杂度为O(length\*wigth),空间复杂度为O(wigth),其实我们可以考虑length和wigth的大小，以较小的为单位压缩dp数组。

代码

public int minPathSum(int[][] grid) {

int length = grid.length;//二维数组行长度

int wigth = grid[0].length;//二维数组宽长度

int[] dp = new int[wigth];//我们以列为单位建立dp数组

for (int i = 0; i < length; i++) {

for (int j = 0; j < wigth; j++) {

if (j == 0) {

if (i == 0)//i=0&&j=0,为起点，dp[j]=1

dp[j] = 1;

else//j=0,为上边界

//new dp[j] = grid[i][j] + old dp[j]

dp[j] = grid[i][j] + dp[j];

}

else {

if (i == 0)//i=0，为左边界

dp[j] = grid[i][j] + dp[j - 1];

else {//i!=0&&j!=0，非边界

int min = Math.min(dp[j], dp[j - 1]);

dp[j] = grid[i][j] + min;

}

}

}

}