

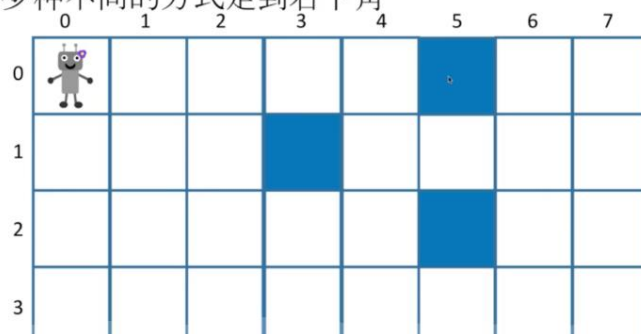
动态规划初探

- 坐标型动态规划
- 序列型动态规划
- 划分性动态规划

坐标型动态规划

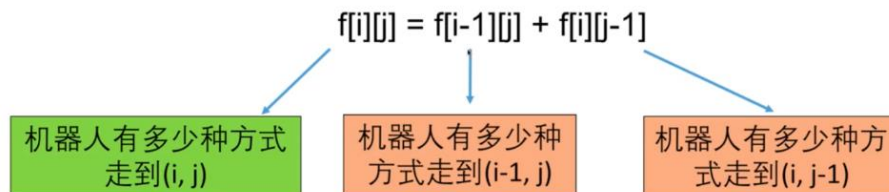
LintCode 115: Unique Paths II

- 题意:
- 给定m行n列的网格，有一个机器人从左上角(0,0)出发，每一步可以向下或者向右走一步
- 网格中有些地方有障碍，机器人不能通过障碍格
- 问有多少种不同的方式走到右下角



题目分析

- 这题和Unique Path非常类似，只是网格中可能有障碍
- 最后一步一定是从左边(i, j-1)或上边(i-1, j)过来
- 状态 $f[i][j]$ 表示从左上角有多少种方式走到格子(i, j)
- 坐标型动态规划：数组下标 $[i][j]$ 即坐标(i, j)



初始条件和边界情况

- $f[i][j]$ = 机器人有多少种方式从左上角走到(i, j)
- 如果左上角(0, 0)格或者右下角(m-1, n-1)格有障碍，直接输出0
- 如果(i, j)格有障碍， $f[i][j] = 0$ ，表示机器人不能到达此格 (0种方式)
- 初始条件： $f[0][0] = 1$

$$f[i][j] = \begin{cases} 0, & \text{如果}(i, j)\text{格有障碍} \\ 1, & i=0 \text{ 且 } j=0 \\ f[i-1][j], & \text{如果 } j=1, \text{ 即第一列} \\ f[i][j-1], & \text{如果 } i=1, \text{ 即第一行} \\ f[i-1][j] + f[i][j-1], & \text{其他} \end{cases}$$

0	1	2	3	4	5	6	7
0	🤖				█		
1			█				
2					█		
3					█		

阶段：
每一个格子都是一个阶段；

决策集：
每一个阶段可选择从上或从左到达；

```

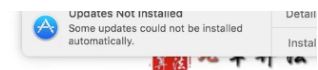
public int uniquePathsWithObstacles(int[][] A) {
    // write your code here
    int m = A.length;
    int n = A[0].length;
    if (A[0][0] == 1 || A[m - 1][n - 1] == 1) {
        return 0;
    }
    int[][] f = new int[m][n];
    f[0][0] = 1;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (A[i][j] == 1) {
                f[i][j] = 0;
                continue;
            } else {
                if (i - 1 >= 0) {
                    f[i][j] += f[i - 1][j];
                }

                if (j - 1 >= 0) {
                    f[i][j] += f[i][j - 1];
                }
            }
        }
    }
    return f[m - 1][n - 1];
}

```

序列型动态规划

LintCode 515 Paint House



- 题意：
- 有一排N栋房子，每栋房子要漆成3种颜色中的一种：红、蓝、绿
- 任何两栋相邻的房子不能漆成同样的颜色
- 第i栋房子染成红色、蓝色、绿色的花费分别是cost[i][0], cost[i][1], cost[i][2]
- 问最少需要花多少钱油漆这些房子
- 例子：
- 输入：
 - N=3
 - Cost = [[14,2,11],[11,14,5],[14,3,10]]
- 输出：
 - 10（第0栋房子蓝色，第1栋房子绿色，第2栋房子蓝色，2+5+3=10）

动态规划组成部分一：确定状态

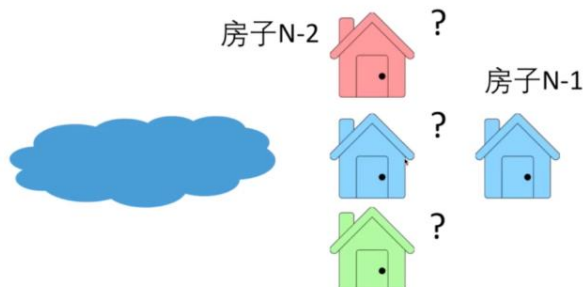
- 最优策略是花费最小的策略
- 最后一步：最优策略中房子N-1一定染成了红、蓝、绿中的一种
- 但是相邻两栋房子不能漆成一种颜色
- 所以如果最优策略中房子N-1是红色，房子N-2只能是蓝色或绿色
- 所以如果最优策略中房子N-1是蓝色，房子N-2只能是红色或绿色
- 所以如果最优策略中房子N-1是绿色，房子N-2只能是红色或蓝色

好复杂。。



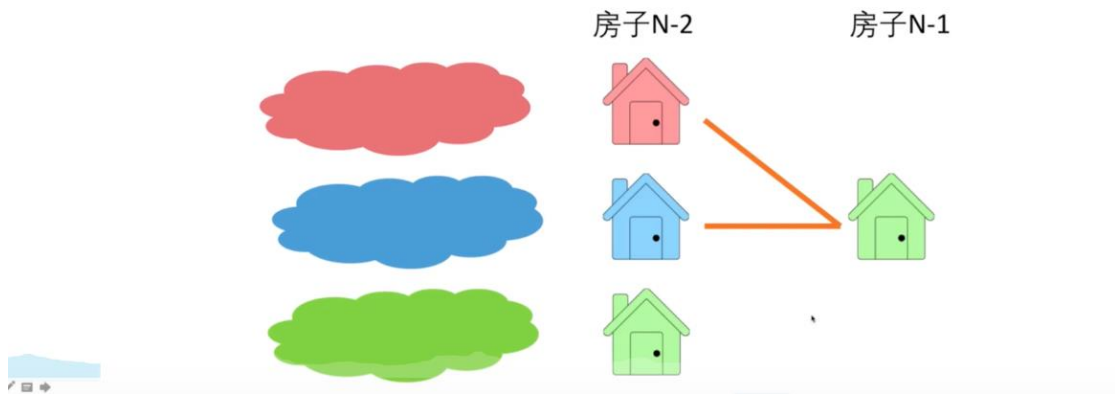
动态规划组成部分一：确定状态

- 如果直接套用以前的思路，记录油漆前N栋房子的最小花费
- 根据套路，也需要记录油漆前N-1栋房子的最小花费
- 但是，前N-1栋房子的最小花费的最优策略中，不知道房子N-2是什么颜色，所以有可能和房子N-1撞色



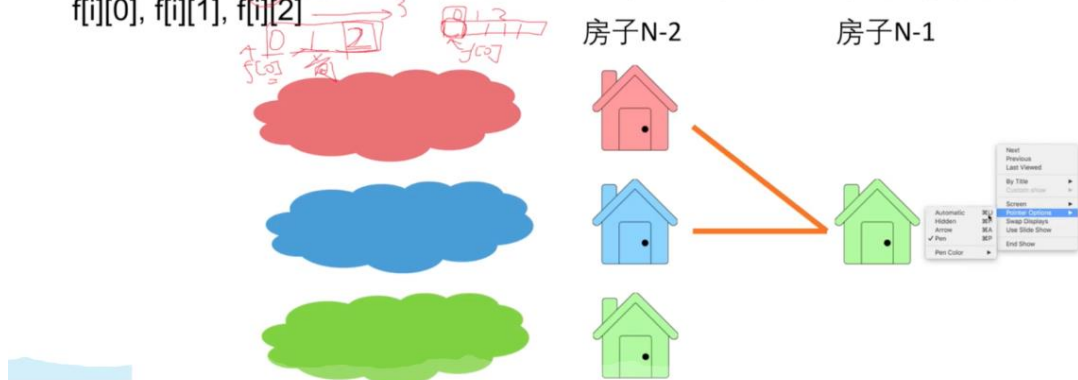
动态规划组成部分一：确定状态

- 不知道房子N-2是什么颜色，就把它记录下来！
- 分别记录油漆前N-1栋房子并且房子N-2是红色、蓝色、绿色的最小花费



子问题

- 求油漆前N栋房子并且房子N-1是红色、蓝色、绿色的最小花费
- 需要知道油漆前N-1栋房子并且房子N-2是红色、蓝色、绿色的最小花费
- 子问题
- 状态：设油漆前i栋房子并且房子i-1是红色、蓝色、绿色的最小花费分别为 $f[i][0]$, $f[i][1]$, $f[i][2]$



动态规划组成部分二：转移方程

- 设油漆前*i*栋房子并且房子*i-1*是红色、蓝色、绿色的最小花费分别为 $f[i][0]$, $f[i][1]$, $f[i][2]$

$$f[i][0] = \min\{f[i-1][1] + \text{cost}[i-1][0], f[i-1][2] + \text{cost}[i-1][0]\}$$

$$f[i][1] = \min\{f[i-1][0] + \text{cost}[i-1][1], f[i-1][2] + \text{cost}[i-1][1]\}$$

$$f[i][2] = \min\{f[i-1][0] + \text{cost}[i-1][2], f[i-1][1] + \text{cost}[i-1][2]\}$$

动态规划组成部分三：初始条件和边界情况

- 设油漆前*i*栋房子并且房子*i-1*是红色、蓝色、绿色的最小花费分别为 $f[i][0]$, $f[i][1]$, $f[i][2]$
- 初始条件： $f[0][0] = f[0][1] = f[0][2] = 0$
 - 即不油漆任何房子的花费
- 无边界情况

动态规划组成部分四：计算顺序

- 设油漆前*i*栋房子并且房子*i-1*是红色、蓝色、绿色的最小花费分别为 $f[i][0]$, $f[i][1]$, $f[i][2]$
- 初始化 $f[0][0]$, $f[0][1]$, $f[0][2]$
- 计算 $f[1][0]$, $f[1][1]$, $f[1][2]$
- ...
- 计算 $f[N][0]$, $f[N][1]$, $f[N][2]$
- 答案是 $\min\{f[N][0], f[N][1], f[N][2]\}$. 时间复杂度 $O(N)$, 空间复杂度 $O(N)$

```

public int minCost(int[][] costs) {
    // write your code here
    int m=costs.length;
    if(m==0 || costs==null){
        return 0;
    }
    int[][] f= new int[m][3];
    f[0][0]=costs[0][0];
    f[0][1]=costs[0][1];
    f[0][2]=costs[0][2];

    for(int i=1;i<m;i++){
        f[i][0]=Math.min(f[i-1][1],f[i-1][2])+costs[i][0];
        f[i][1]=Math.min(f[i-1][0],f[i-1][2])+costs[i][1];
        f[i][2]=Math.min(f[i-1][1],f[i-1][0])+costs[i][2];
    }
    return Math.min(Math.min(f[m-1][0],f[m-1][1]),f[m-1][2]);
}

```

小结

- 序列型动态规划：...前*i*个...最小/方式数/可行性
- 在设计动态规划的过程中，发现需要知道油漆前*N-1*栋房子的最优策略中，房子*N-2*的颜色
- 如果只用*f[N-1]*，将无法区分
- 解决方法：记录下房子*N-2*的颜色
 - 在房子*N-2*是红/蓝/绿色的情况下，油漆前*N-1*栋房子的最小花费
- 问题迎刃而解
- **序列+状态**

划分型动态规划

LintCode 512 Decode Ways

- 题意：
- 有一段由A-Z组成的字母串信息被加密成数字串
- 加密方式为：A→1, B→2, ..., Z→26
- 给定加密后的数字串S[0...N-1]，问有多少种方式解密成字母串
- 例子：
- 输入：
 - 12
- 输出：
 - 2 (AB 或者 L)

动态规划组成部分一：确定状态

- 解密数字串即划分成若干段数字，每段数字对应一个字母
- 最后一步（最后一段）：^{1/2}对应一个字母
- A, B, ..., 或 Z
- 这个字母加密时变成 1, 2, ..., 或 26

0	1	2		N-4	N-3	N-2	N-1
8	2	3	...	1	3	1	2

假设 100 种解密方式

B

动态规划组成部分一：确定状态

- 解密成为字母串
- 最后一步：一定有最后一个字母
- A, B, ..., 或 Z
- 这个字母加密时变成 1, 2, ..., 或 26

0	1	2		N-4	N-3	N-2	N-1
8	2	3	...	1	3	1	2

假设 50 种解密方式

L

动态规划组成部分一：确定状态

- 解密成为字母串
- 最后一步：一定有最后一个字母
- A, B, ..., 或 Z
- 这个字母加密时变成 1, 2, ..., 或 26

0	1	2		N-4	N-3	N-2	N-1
8	2	3	...	1	3	1	2

一共 $100 + 50 = 150$ 种解密方式

****阶段：****
每个数字作为结尾作为一个阶段，有 n 个数字就有 n 个阶段。

****决策集：****
决策一：当前阶段的作为结尾的数字划分出两位数字；
决策二：当前阶段的作为结尾的数字划分出一位数字；

子问题

****状态表示：****
 $f[i]$ 表示数字串前 i 个数字解密成字符串的方式数。

- 设数字串长度为 N
- 要求数字串前 N 个字符的解密方式数
- 需要知道数字串前 $N-1$ 和 $N-2$ 个字符的解密方式数
- 子问题
- 状态：设数字串 S 前 i 个数字解密成字母串有 $f[i]$ 种方式

动态规划组成部分二：转移方程

- 设数字串 S 前 i 个数字解密成字母串有 $f[i]$ 种方式

$$f[i] = f[i-1] \mid S[i-1] \text{ 对应一个字母} + f[i-2] \mid S[i-2]S[i-1] \text{ 对应一个字母}$$

数字串 S 前 i 个数字解密成字母串的方式数

数字串 S 前 $i-1$ 个数字解密成字母串的方式数

数字串 S 前 $i-2$ 个数字解密成字母串的方式数

动态规划组成部分三：初始条件和边界情况

- 设数字串 S 前 i 个数字解密成字母串有 $f[i]$ 种方式
- 初始条件： $f[0] = 1$ ，即空串有 1 种方式解密
— 解密成空串
- 边界情况：如果 $i = 1$ ，只看最后一个数字

动态规划组成部分四：计算顺序

- $f[0], f[1], \dots, f[N]$
- 答案是 $f[N]$
- 时间复杂度 $O(N)$ ，空间复杂度 $O(N)$

```

public int numDecodings(String s) {
    // write your code here
    char[] ss = s.toCharArray();
    int n = ss.length;
    if(n==0){
        return 0;
    }
    int[] f = new int[n+1];
    f[0]=1;
    for(int i=1;i<=n;i++){ // 前i个字符，不将第i个字符计算在内
        int num1=ss[i-1]-'0';
        if(1<=num1 && num1<=9){
            f[i] +=f[i-1];
        }

        if(i>=2){
            int num2=(ss[i-2]-'0')*10 + (ss[i-1]-'0');
            if(10<=num2 && num2<=26){
                f[i] +=f[i-2];
            }
        }
    }
    return f[n];
}

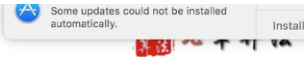
```

坐标型动态规划

- 最简单的动态规划类型
- 给定一个序列或网格
- 需要找到序列中某个/些子序列或网格中的某条路径
 - 某种性质最大/最小
 - 计数
 - 存在性
- 动态规划方程 $f[i]$ 中的下标 i 表示以 a_i 为结尾的满足条件的子序列的性质， $f[i][j]$ 中的下标 i, j 表示以格子 (i, j) 为结尾的满足条件的路径的性质
 - 最大值/最小值
 - 个数
 - 是否存在
- 坐标型动态规划的初始条件 $f[0]$ 就是指以 a_0 为结尾的子序列的性质

坐标型动态规划

LintCode 397 最长连续单调子序列



- 题意：
- 给定 $a[0], \dots, a[n-1]$
- 找到最长的连续子序列 $i, i+1, i+2, \dots, j$, 使得 $a[i] < a[i+1] < \dots < a[j]$, 或者 $a[i] > a[i+1] > \dots > a[j]$, 输出长度 $j-i+1$
- 例子：
- 输入： $[5, 1, 2, 3, 4]$
- 输出：4 (子序列 $1, 2, 3, 4$)

简化



- 首先，对于 $a[i] > a[i+1] > \dots > a[j]$, 可以将整个 a 序列倒过来，就变成求最长连续上升子序列了
- 所以，只需要考虑找到最长的 $a[i] < a[i+1] < \dots < a[j]$
- 可以从每个 $a[i]$ 开始，一直向后延伸找到最长的连续上升序列
- 最差情况下，对于长度为 N 的序列，需要计算 $O(N^2)$ 步：
 $-0, 1, 2, \dots, N-1$

动态规划组成部分一：确定状态



- 最后一步：对于最优的策略，一定有最后一个元素 $a[j]$
- 第一种情况：最优策略中最长连续上升子序列就是 $\{a[j]\}$, 答案是1

先考虑简单情况



- 第二种情况，子序列长度大于1，那么最优策略中 $a[j]$ 前一个元素肯定是 $a[j-1]$. 这种情况一定是 $a[j-1] < a[j]$



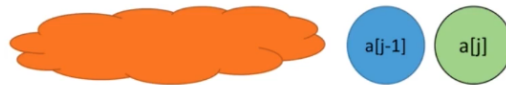
- 因为是最优策略，那么它选中的以 $a[j-1]$ 结尾的连续上升子序列一定是最长的

阶段：
以每个数字作为结尾当前一个阶段，有 N 个数字就是 N 个阶段。

决策集：
每个作为结尾的数字与之前的数字比较一次都是一次决策。

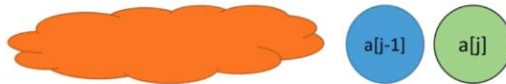
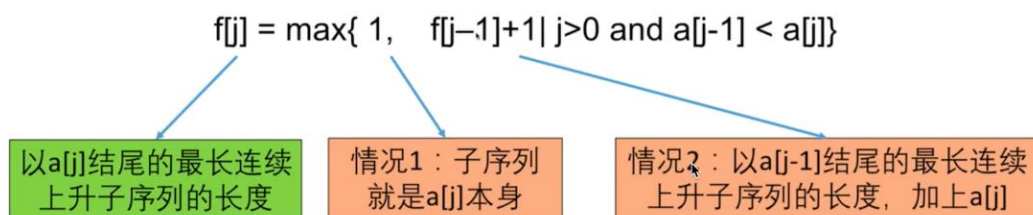
子问题

- 要求以 $a[j-1]$ 结尾的最长连续上升子序列
- 本来是求以 $a[j]$ 结尾的最长连续上升子序列
- 化为子问题
- 状态：设 $f[j]$ = 以 $a[j]$ 结尾的最长连续上升子序列的长度

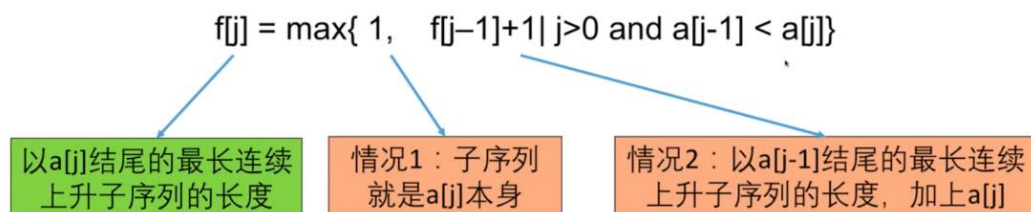


动态规划组成部分二：转移方程

- $f[j]$ = 以 $a[j]$ 结尾的最长连续上升子序列的长度



动态规划组成部分三：初始条件和边界情况



- 情况2必须满足：
 - $j>0$, 即 $a[j]$ 前面至少还有一个元素
 - $a[j] > a[j-1]$, 满足单调性
- 初始条件：空

动态规划组成部分四：计算顺序

- $f[j]$ = 以 $a[j]$ 结尾的最长连续上升子序列的长度
- 计算 $f[0], f[1], f[2], \dots, f[n-1]$
- 和硬币组合题不一样的是，最终答案并不一定是 $f[n-1]$
- 因为我们不知道最优策略中最后一个元素是哪个 $a[j]$
- 所以答案是 $\max\{f[0], f[1], f[2], \dots, f[n-1]\}$
- 算法时间复杂度 $O(n)$ ，空间复杂度 $O(n)$

思考：如何做到空间复杂度 $O(1)$

LintCode 110 Minimum Path Sum

- 题意：
- 给定 m 行 n 列的网格，每个格子 (i, j) 里都有一个非负数 $A[i][j]$
- 求一个从左上角 $(0, 0)$ 到右下角的路径，每一步只能向下或者向右走一步
- 使得路径上的格子里的数字之和最小
- 输出最小数字和

	0	1	2	3	4
0	1	5	7	6	8
1	4	7	4	4	9
2	10	3	2	3	2

动态规划组成部分一：确定状态

- 最值型动态规划
- 和 **Unique Path** 一样，无论用何种方式到达右下角，总有最后一步：
 - 向右 或者 向下
- 右下角坐标设为 $(m-1, n-1)$
- 那么前一步一定是在 $(m-2, n-1)$ 或者 $(m-1, n-2)$

	0	1	2	3	4
0	1	5	7	6	8
1	4	7	4	4	9
2	10	3	2	3	2

Diagram illustrating the grid with arrows indicating the path from (0,0) to (2,4):

- From (0,0) to (0,1): right arrow
- From (0,1) to (1,1): down arrow
- From (1,1) to (1,2): right arrow
- From (1,2) to (2,2): down arrow
- From (2,2) to (2,3): right arrow
- From (2,3) to (2,4): right arrow

动态规划组成部分一：确定状态

- 最优策略的路径总和数字最小
 - 若倒数第二步在(m-2, n-1)，则前面一定是从(0,0)到达(m-2, n-1)总和最小的路径
 - 若倒数第二步在(m-1, n-2)，则前面一定是从(0,0)到达(m-1, n-2)总和最小的路径

	0	1	2	3	4
0	1	5	7	6	8
1	4	7	4	4	9
2	10	3	2	3	2

Diagram showing a 3x5 grid with values. A path is highlighted from (0,0) to (2,4) with arrows: (0,0) → (1,0) → (2,0) → (2,1) → (2,2).

子问题

- 要求从左上角走到(m-1, n-2)的路径的最小数字总和以及走到(m-2, n-1)的路径的最小数字总和
- 原题要求有从左上角走到(m-1, n-1)的路径的最小数字总和
- 子问题
- 状态：
 - 设从(0, 0)走到(i, j)的路径最小数字总和为 $f[i][j]$

	0	1	2	3	4
0	1	5	7	6	8
1	4	7	4	4	9
2	10	3	2	3	2

Diagram showing a 3x5 grid with values. A path is highlighted from (0,0) to (2,4) with arrows: (0,0) → (1,0) → (2,0) → (2,1) → (2,2).

动态规划组成部分二：转移方程

- 设从(0, 0)走到格子(i, j)的路径的最小数字总和是 $f[i][j]$

$$f[i][j] = \min\{f[i-1][j], f[i][j-1]\} + A[i][j]$$

从(0, 0)走到格子(i, j)的最小路径数字总和

从(0, 0)走到格子(i-1, j)的最小路径数字总和

从(0, 0)走到格子(i, j-1)的最小路径数字总和

格子(i, j)的数字

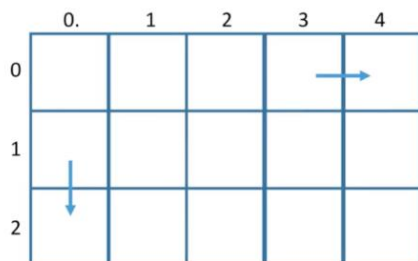
	0	1	2	3	4	5	6	7
0								
1								
2								
3								

Diagram showing a 4x8 grid with a robot at (3, 7) and arrows indicating movement from (3, 6) and (2, 7).

动态规划组成部分三：初始条件和边界情况

九章算法

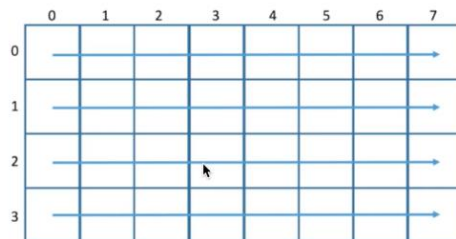
- 初始条件： $f[0][0] = A[0][0]$
- 边界情况： $i = 0$ 或 $j = 0$ ，则前一步只能有一个方向过来



动态规划组成部分四：计算顺序

九章算法

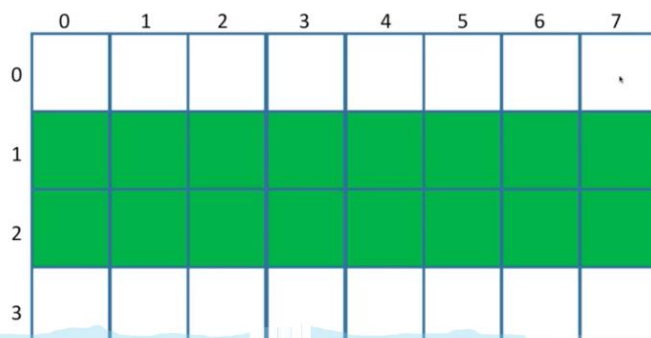
- $f[0][0] = 1$
- 计算第0行： $f[0][0], f[0][1], \dots, f[0][n-1]$
- 计算第1行： $f[1][0], f[1][1], \dots, f[1][n-1]$
- ...
- 计算第m-1行： $f[m-1][0], f[m-1][1], \dots, f[m-1][n-1]$
- $f[i][j] = f[i-1][j] + f[i][j-1]$
- 时间复杂度（计算步数）： $O(MN)$ ，空间复杂度（数组大小）： $O(MN)$



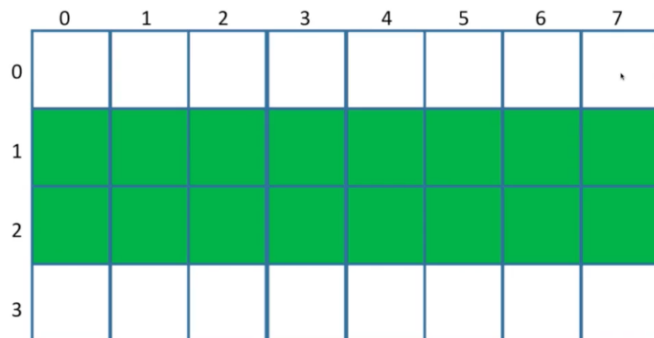
空间优化

九章算法

- $f[i][j] = f[i-1][j] + f[i][j-1]$
- 计算第i行时，只需要第i行和第i-1行的f



- 所以，只需要保存两行的f值： $f[i][0..n-1]$ 和 $f[i-1][0..n-1]$
- 用滚动数组实现



	0	1	2	3	4	5	6	7
0								
1								
2								
3								

- 开数组时，只开 $f[0][0..n-1]$ 和 $f[1][0..n-1]$
- 计算 $f[0][0], \dots, f[0][n-1]$, 计算 $f[1][0], \dots, f[1][n-1]$
- 计算 $f[2][0..n-1]$ 时，开 $f[2][0..n-1]$ ，删掉 $f[0][0..n-1]$ ，因为已经不需要 $f[0][0..n-1]$ 的值了
- 计算 $f[3][0..n-1]$ 时，开 $f[3][0..n-1]$ ，删掉 $f[1][0..n-1]$ ，因为已经不需要 $f[1][0..n-1]$ 的值了

- 实际操作时，可以不用每次开数组，而是用滚动法
- 计算 $f[0][0], \dots, f[0][n-1]$, 计算 $f[1][0], \dots, f[1][n-1]$
- 计算 $f[2][0..n-1]$ 时，把值写在 $f[0][0..n-1]$ 的数组里
- 同理， $f[3][0..n-1]$ 写在 $f[1][0..n-1]$ 的数组里
- 最后 $f[m-1][n-1]$ 存储在 $f[0][n-1]$ （或者 $f[1][n-1]$ ）里，直接输出

知识点

对于网格上的动态规划，如果 $f[i][j]$ 只依赖于本行的 $f[i][x]$ 与前一行的 $f[i-1][y]$ ，那么就可以采用滚动数组的方法压缩空间。空间复杂度 $O(N)$

	0	1	2	3	4	5	6	7
0								
1								
2								
3								

知识点

如果网格行数少列数多（大胖子网格），那么就可以逐列计算，滚动数组的长度为行数，空间复杂度 $O(M)$

	0	1	2	3	4	5	6	7
0								
1								
2								
3								

常用滚动数组控制写法：

```
int old = 1, now = 0;
```

每一次循环都要交换：

```
old = now;
```

```
now = 1 - now;
```

LintCode 553 Bomb Enemy

- 题意：
- 有一个 $M \times N$ 的网格，每个格子可能是空的，可能有一个敌人，可能有一堵墙
- 只能在某个空格子里放一个炸弹，炸弹会炸死所有同行同列的敌人，但是不能穿透墙
- 最多能炸死几个敌人

- 例子：
- 输入：如图
- 输出：3

	0	1	2	3
0	0	E	0	0
1	E	0	W	E
2	0	E	0	0

题目分析



- 每个炸弹可以往四个方向传播爆炸力
- 我们可以分析一个方向，然后举一反三
- 即如果在一个空地放一个炸弹，最多向上能炸死多少敌人

• 可以直接枚举，即向上枚举到碰到墙为止
碰到墙就枚举停止

- 时间复杂度 $O(MN*M)$
- 用动态规划思想加速

	0	1	2	3
0	0	E	0	0
1	E	W		E
2	0	E	0	0

动态规划组成部分一：确定状态



- 我们假设有敌人或有墙的格子也能放炸弹
 - 有敌人的格子：格子上的敌人被炸死，并继续向上爆炸
 - 有墙的格子：炸弹不能炸死任何敌人
- 在 (i, j) 格放一个炸弹，它向上能炸死的敌人数是：
 - (i, j) 格为空地： $(i-1, j)$ 格向上能炸死的敌人数
 - (i, j) 格为敌人： $(i-1, j)$ 格向上能炸死的敌人数+1
 - (i, j) 格为墙：0

	0	1	2	3
0	0	E	0	0
1	E	W		E
2	0	E	0	0

子问题



- 需要知道 $(i-1, j)$ 格放一个炸弹向上能炸死的敌人数
- 原来要求 (i, j) 格放一个炸弹向上能炸死的敌人数
- 子问题
- 状态：

– $Up[i][j]$ 表示 (i, j) 格放一个炸弹向上能炸死的敌人数

	0	1	2	3
0	0	E	0	0
1	E	W		E
2	0	E	0	0


动态规划组成部分三：初始条件和边界情况

九章算法

- 设 $Up[i][j]$ 表示 (i, j) 格放一个炸弹向上能炸死的敌人数

$$Up[i][j] = \begin{cases} Up[i-1][j], & \text{如果}(i, j)\text{格是空地} \\ Up[i-1][j] + 1, & \text{如果}(i, j)\text{格是敌人} \\ 0, & \text{如果}(i, j)\text{格是墙} \end{cases}$$

- 初始条件：第0行的 Up 值和格子内容相关
 - $Up[0][j] = 0$, 如果 $(0, j)$ 格不是敌人
 - $Up[0][j] = 1$, 如果 $(0, j)$ 格是敌人

	0	1	2	3
0	0	E	0	0
1	E		W	E
2	0	E	0	0

动态规划组成部分四：计算顺序

九章算法




- 逐行计算
 - $Up[0][0], Up[0][1], \dots, Up[0][n-1]$
 - $Up[1][0], Up[1][1], \dots, Up[1][n-1]$
 - ...
 - $Up[m-1][0], Up[m-1][1], \dots, Up[m-1][n-1]$
- 时间复杂度 $O(MN)$, 空间复杂度 $O(MN)$

坐标型动态规划总结

九章算法

- 给定输入为序列或者网格/矩阵
- 动态规划状态下标为序列下标 i 或者网格坐标 (i, j)
 - $f[i]$: 以第 i 个元素结尾的某种性质
 - $f[i][j]$: 到格子 (i, j) 的路径的性质
- 初始化设置 $f[0]$ 的值/ $f[0][0..n-1]$ 的值
- 二维空间优化：如果 $f[i][j]$ 的值只依赖于当前行和前一行，则可以用滚动数组节省空间

	0	1	2	3	4	5	6	7
0								
1								
2								
3								

	1	2	3	4	5	6	7	8
1								
2								
3								
4								

LintCode 664 Counting Bits

九章算法

- 题意：
- 给定N，要求输出0, 1, ..., N的每个数的二进制表示里的1的个数
- 例子：
- 输入：5
- 输出：[0, 1, 1, 2, 1, 2]
- 0 : 0
- 1 : 1
- 2 : 10
- 3 : 11
- 4 : 100
- 5 : 101

题目分析

九章算法

- 对于每个数 $0 \leq i \leq N$ ，直接求i的二进制表示里有多少个1
- 二进制表示算法：
 - 第一步: $i \bmod 2$ 是最低位的bit
 - 第二步: $i \leftarrow \text{floor}(i/2)$ ，如果 $i=0$ ，结束，否则回到第一步
- 时间复杂度： $O(N \log N)$
 - 2个数有1位二进制
 - 2个数有2位二进制
 - 4个数有3位二进制
 - 8个数有4位二进制
 - ...
 - 大约 $N/2$ 个数有 $\log_2 N$ 位二进制

子问题

九章算法

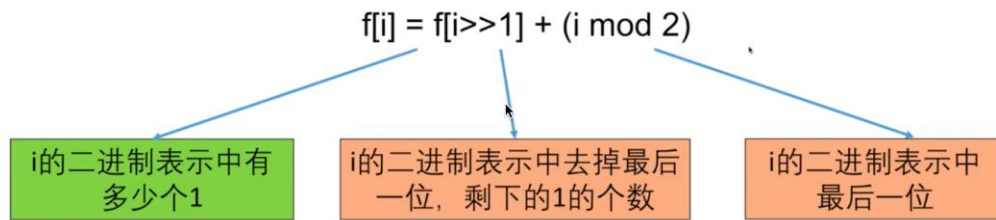
- 要求N的二进制表示中有多少1
- 在N的二进制去掉最后一位 $N \bmod 2$ ，设新的数是 $Y=(X \gg 1)$ (右移一位)
- 要知道Y的二进制表示中有多少1
- 子问题
- 状态：设 $f[i]$ 表示i的二进制表示中有多少个1

知识点：和位操作相关的
动态规划一般用值作状态

动态规划组成部分二：转移方程

九章算法

- 设 $f[i]$ 表示 i 的二进制表示中有多少个1



动态规划组成部分三：初始条件和边界情况

九章算法

- 设 $f[i]$ 表示 i 的二进制表示中有多少个1
- $f[i] = f[i >> 1] + (i \bmod 2)$
- 初始条件： $f[0] = 0$

动态规划组成部分四：计算顺序

九章算法

- $f[0], f[1], f[2], \dots, f[N]$
- 时间复杂度 $O(N)$
- 空间复杂度 $O(N)$