Count the paths

paths 拆分成 求A、B到终点路径有多少步的子问题（分治，**几种选择就分几个子问题**）

paths (start, end) =

paths (A, end) + paths (B, end)

= =

paths (D, end) + paths (C, end) paths (C, end) + paths (E, end)

```
int countPaths (boolean[][]grid, int row, int col) {
    if (!validSquare(grid, row, col)) return 0;
    if (isAtEnd(grid, row, col)) return 1;
    return countPaths (grid, row + 1, col) + countPaths (grid, row, col + 1);
}
```

递推: 从终点往起点走

动态规划

优化重复计算，利用缓存(Hash) 实现记忆化搜索(Memorization) ------- e.g fibnacci

只要写递归，就直接写 for 循环来实现**自底向**上的递推(竞赛选手做法)

复杂点的递归会是多维度的比如二维、三维数据。
它中间会有所谓的取舍最优子结构

有时候结果要取累加、有时候结果要取最大或最小值

动态规划关键点

1. 最优子结构 opt[n] = best_of(opt[n-1], opt[n-2], ...)

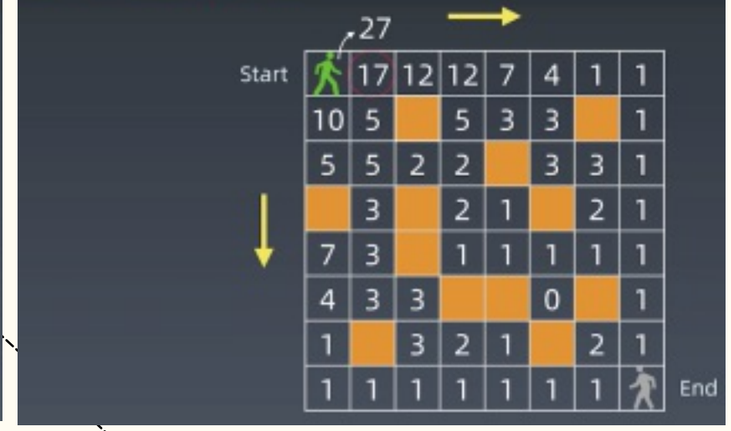
2. 储存中间状态：opt[i]

3. 递推公式（美其名曰：状态转移方程或者 DP方程）

Fib: opt[i] = opt[n-1] + opt[n-2]

二维路径：opt[i,j] = opt[i+1][j] + opt[i][j+1] (且判断a[i,j]是否空地)

Count the paths



. DP

a. 重复性（分治）

b. 定义状态数组

c. DP方程

字符串定义为DP时，需要将字符串转换为数组来定义状态

两个字符串的比较就升维度，变二维
比较的时候可以从后往前比会更自
定义状态的时候，数组注意不要越界

递推公式困难的话，可以想一下自底向上的思路

小结
1.打破自己思维习惯，形成机械思维 (找重复性, 计算机只会 if else)
2.理解复杂逻辑的关键（定义状态数组, DP方程）
3.也是职业进阶的要领(不要人肉递归，也就是当管理者不要亲力亲为)

MIT五步DP
1. define subproblems
2. guess(part of solution)
3. relate subproblem solutions
4. recurse & memorize
5. solve original problem

https://www.bilibili.com/video/av53233912?from=search&seid=2847395688604491997