

1 人肉递归低效、很累
2 找到最近最简方法，将其拆解成可重复解决的问题
3 数学归纳法思维（抵制人肉递归的诱惑）

本质：寻找重复性 -> 计算机指令集

人肉递归：画递归树

动态规划

"simplifying a complicated problem by breaking it down into simpler sub-problems"(in a recursive manner)

分治 + 最优子结构

动态规划和递归或分治没有根本上的区别（关键看有无最优子结构）

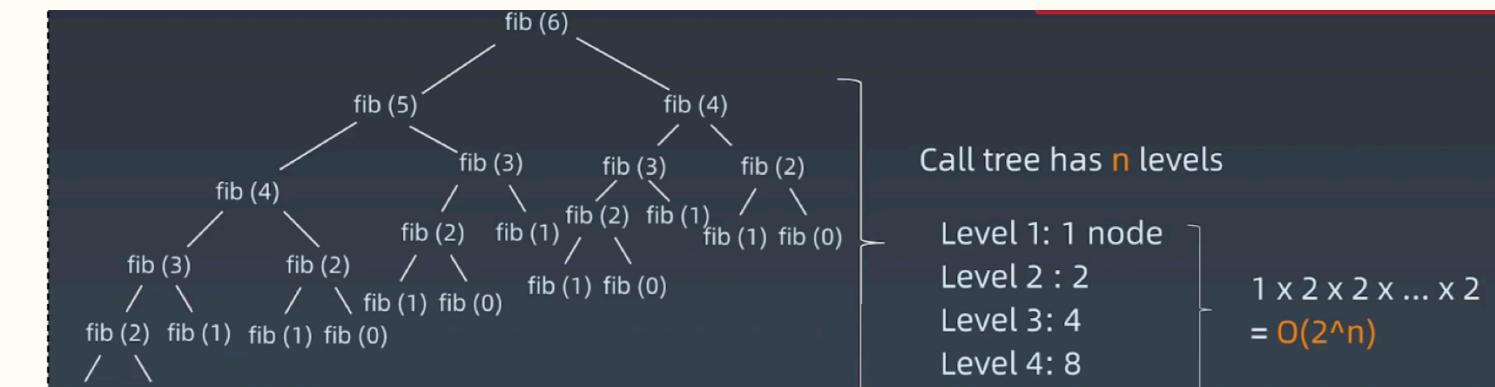
共性：找到重复子问题

差异性：最优子结构，中途可以淘汰次优解

实战：

1 斐波那契数列

1 递归，时间复杂度指数级



```
def fib(n):
    return n if n <= 1 else fib(n - 1) + fib(n - 2)
```

直接递归
时间复杂度: $O(2^n)$

加缓存

两种方法：
1 利用python的
lru_cache缓存
2 构建一个数组来存储值

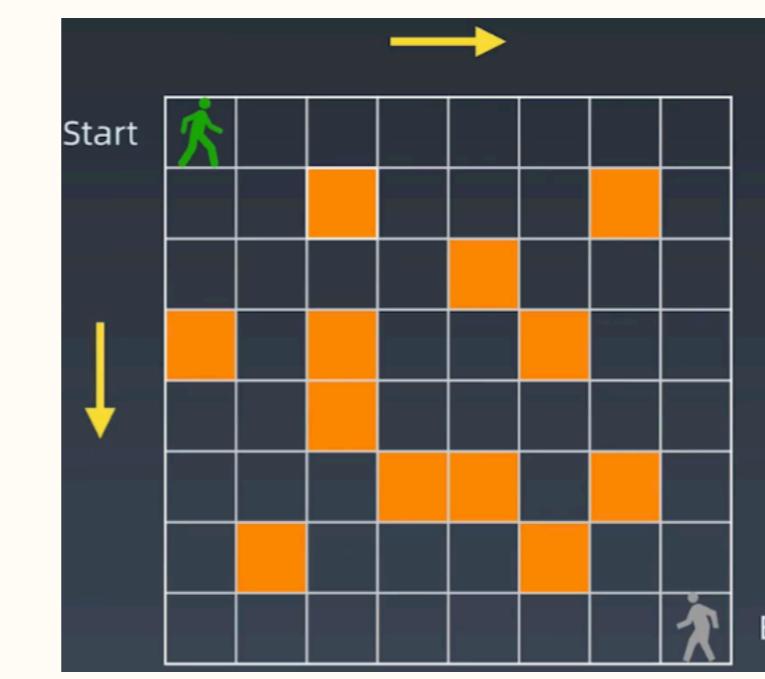
时间复杂度: $O(n)$

这是自顶向下的方法：
即要求fib(6),
fib(6)=fib(5)+fib(4)
=fib(4)+fib(4)+fib(3)
+fib(2)..
print(fib(66))

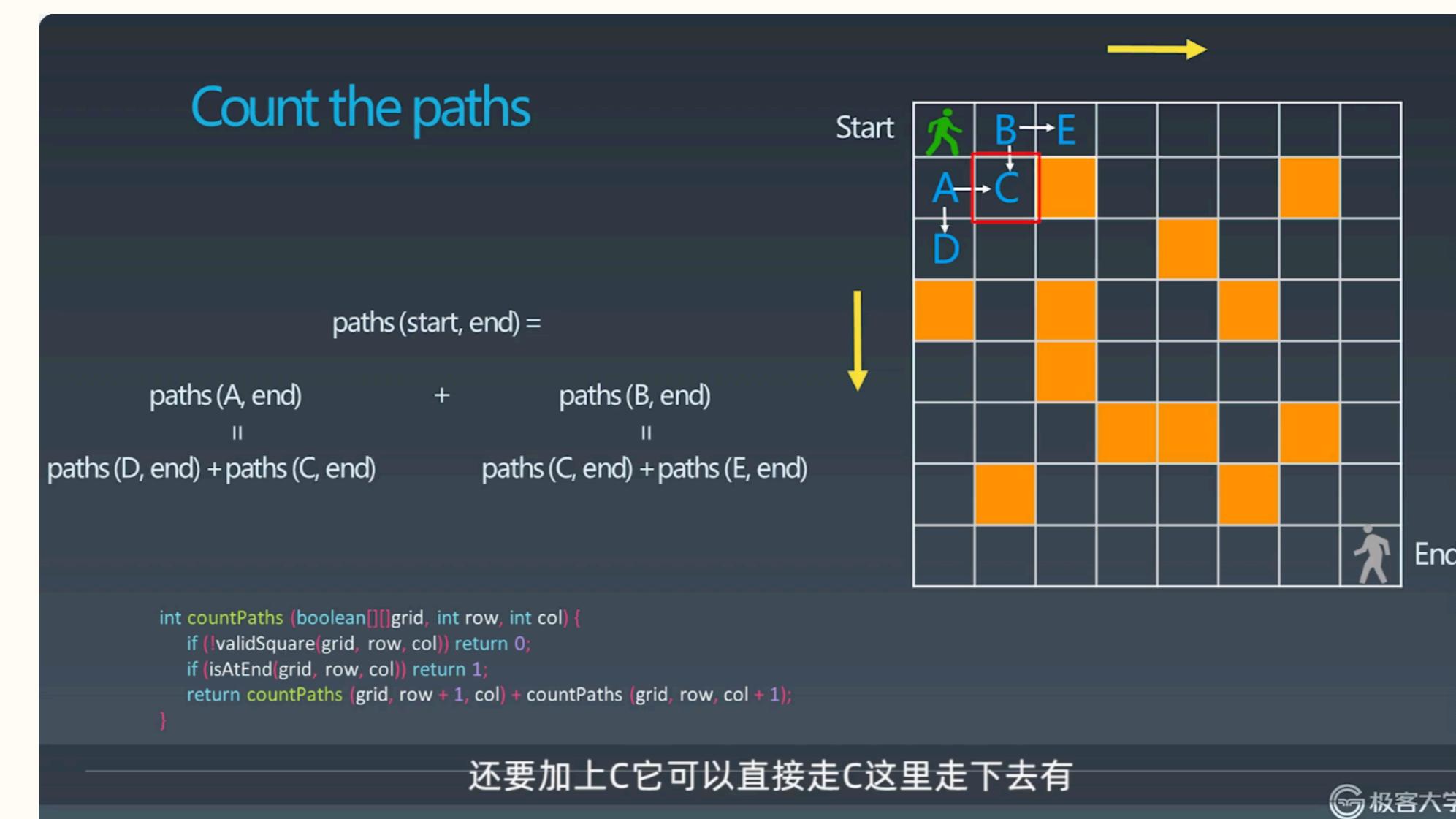
```
def fib(n):
    a = [0] * (n+1)
    a[0], a[1] = 0, 1
    for i in range(2, n+1):
        a[i] = a[i - 1] + a[i - 2]
    return a[n]
print(fib(10))
```

这是自底向上的方法：
即 $a[2] = a[1] + a[0]$
 $a[3] = a[2] + a[1]$
...
 $a[10] = a[9] + a[8]$
最后输出 $a[10]$

2 路径计数

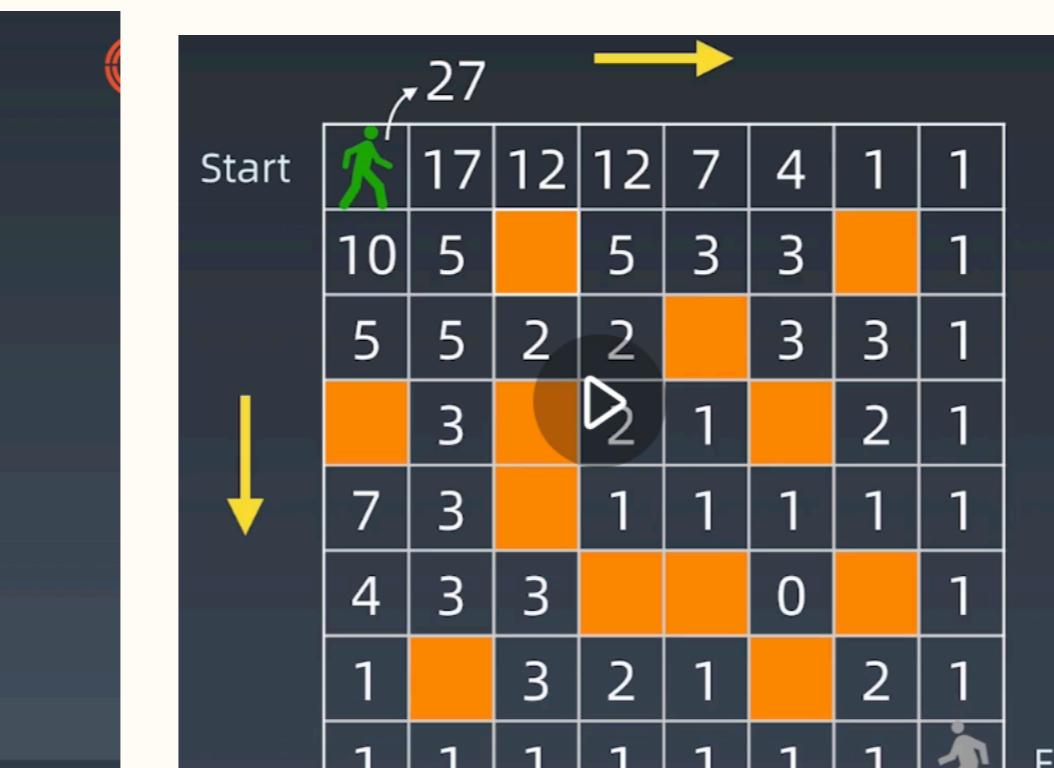


从 start 走向 end，只能是向右和向下走，黄色的地方是障碍物，不能走，问，有多少种走法



状态转移方程 (DP方程)

```
opt[i, j] = opt[i + 1, j] + opt[i, j + 1]
完整逻辑：
if a[i, j] = '空地' :
    opt[i, j] = opt[i + 1, j] + opt[i, j + 1]
else:
    opt[i, j] = 0
就是 f(n) = f(n-1) + f(n-2)
```



64. 最小路径和

难度 中等 649

给定一个包含非负整数的 $m \times n$ 网格，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。

说明：每次只能向下或者向右移动一步。

示例：

输入：
[[1,3,1],
[1,5,1],
[4,2,1]]
输出：7
解释：因为路径 1->3->1->1 的总和最小。

解法一：

```
def minPathSum(self, grid):
    """
    :type grid: List[List[int]]
    :rtype: int
    """
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if i == j == 0: continue
            elif i == 0: grid[i][j] = grid[i][j - 1] + grid[i][j]
            elif j == 0: grid[i][j] = grid[i - 1][j] + grid[i][j]
            else: grid[i][j] = min(grid[i - 1][j], grid[i][j - 1]) + grid[i][j]
    return grid[-1][-1]
```

这是 MxN 网格，
dp[i][j] 表示从 (0,0) 到 (i,j) 的路径和。
如果 (i,j) 是障碍物，则 dp[i][j] = float('inf')。
如果 (i,j) 是终点，则 dp[i][j] = 0。
遍历所有单元格，更新 dp[i][j] 为 min(dp[i-1][j], dp[i][j-1]) + grid[i][j]。
返回 dp[M-1][N-1]。

时间复杂度: $O(M \cdot N)$

空间复杂度: $O(1)$

New Note

解法二：我觉得想法很好玩

```
def minPathSum(grid):
    dp = [float('inf')] * (len(grid[0]) + 1)
    dp[1] = 0
    for row in grid:
        for idx, num in enumerate(row):
            dp[idx + 1] = min(dp[idx], dp[idx + 1]) + num
            print("idx:{} num:{} ".format(idx, num))
    return dp[-1]
```

这是 MxN 网格，
dp[i][j] 表示从 (0,0) 到 (i,j) 的路径和。
如果 (i,j) 是障碍物，则 dp[i][j] = float('inf')。
如果 (i,j) 是终点，则 dp[i][j] = 0。
遍历所有单元格，更新 dp[i][j] 为 min(dp[i-1][j], dp[i][j-1]) + num。
返回 dp[-1]。

时间复杂度: $O(M \cdot N)$

空间复杂度: $O(N)$

62. 不同路径

难度 中等 660

一个机器人位于一个 $m \times n$ 网格的左上角（起始点在下图中标记为“Start”）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。

问总共有多少条不同的路径？



动态规划关键点

极客大学

1. 最优子结构 $opt[n] = \text{best_of}(opt[n-1], opt[n-2], \dots)$

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: $opt[i] = opt[i-1] + opt[i-2]$

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)

类比上一题的解法一，时间复杂度 $O(M \cdot N)$ ，空间复杂度 $O(M \cdot N)$

```
def uniquePaths(self, m, n):
    dp = [[1]*n] + [[1]+[0]*(n-1) for _ in range(m-1)]
    for i in range(1, m):
        for j in range(1, n):
            dp[i][j] = dp[i-1][j] + dp[i][j-1]
    return dp[-1][-1]
```

类比上一题的解法二，时间复杂度 $O(M \cdot N)$ ，空间复杂度 $O(N)$

```
def uniquePaths(self, m, n):
    cur = [1] * n
    for i in range(1, m):
        for j in range(1, n):
            cur[j] += cur[j-1]
    return cur[-1]
```

例如，上图是一个 7×3 的网格。有多少可能的路径？

不同路径II：如同老师讲的例题