动态规划
Dynamic Programming

本节内容

1. 递推！而递归+记忆化是少数情况

2. 状态的定义 fib[n]

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

4. 状态转移方程（DP方程）

要点

1. 递推！而递归+记忆化是少数情况

2. 状态的定义 fib[n]

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

4. 状态转移方程（DP方程）

- Fibonacci array

- 0, 1, 1, 2, 3, 5, 8, 13, 21, …

- F[n] = F[n-1] + F[n-2]

N → R

MEMOIZATION + DYNAMIC PROGRAMMING

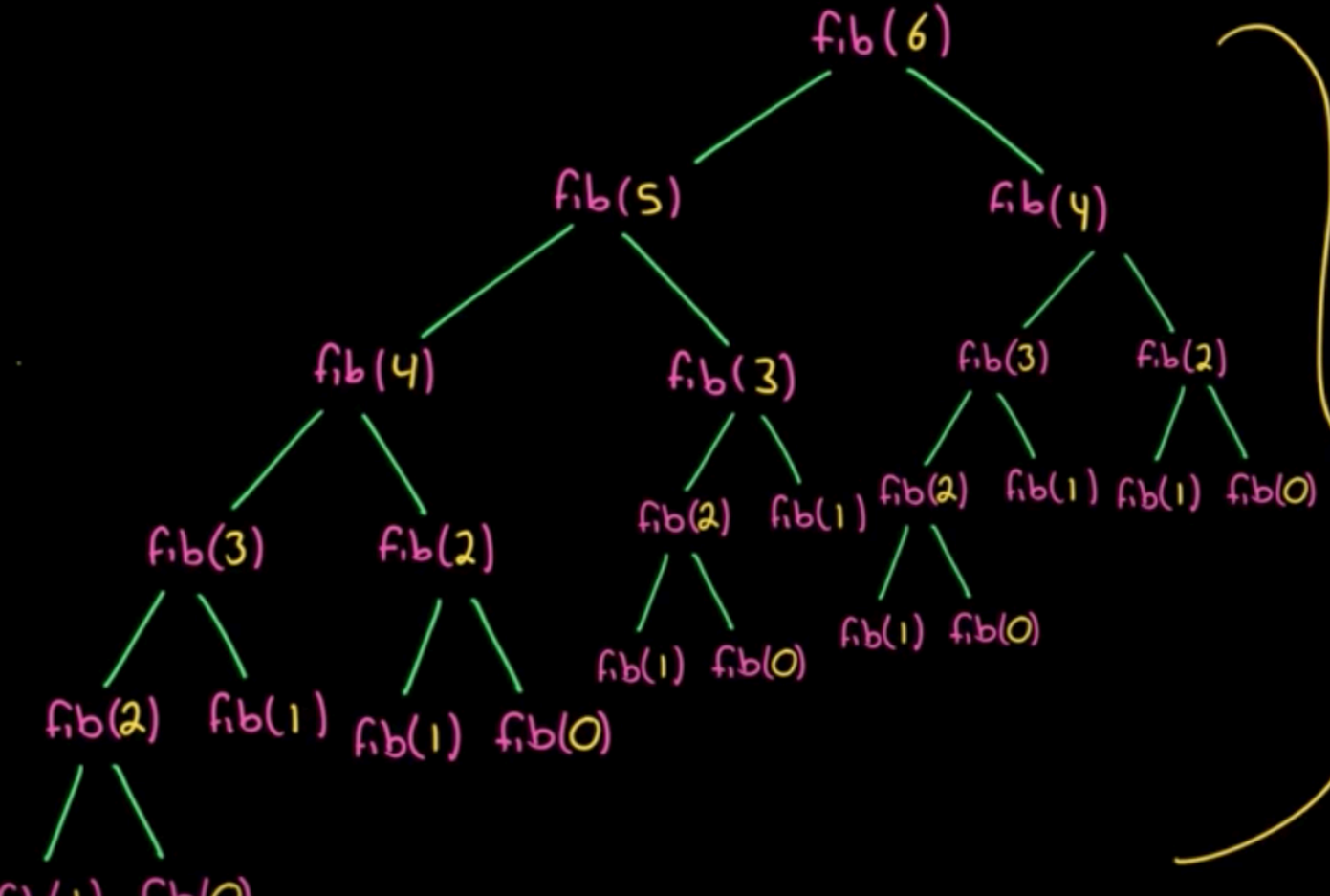
S R

$fib(n) = fib(n-1) + fib(n-2)$

$fib(0) = 0$

$fib(1) = 1$

```
int fib(int n) {
    if (n<=0) {
        return 0;
    } else if (n==1) {
        return 1;
    } else {
        return fib(n-1)+fib(n-2);
    }
}
```



call tree has n levels

Level 1: 1 node

Level 2: 2

Level 3: 4

Level 4: 8

$1 \times 2 \times 2 \times \dots \times 2 = O(2^n)$

```
int fib(int n) {
   return n <= 1 ? n : fib(n - 1) + fib(n - 2);
}
```

N → R

MEMOIZATION + DYNAMIC PROGRAMMING

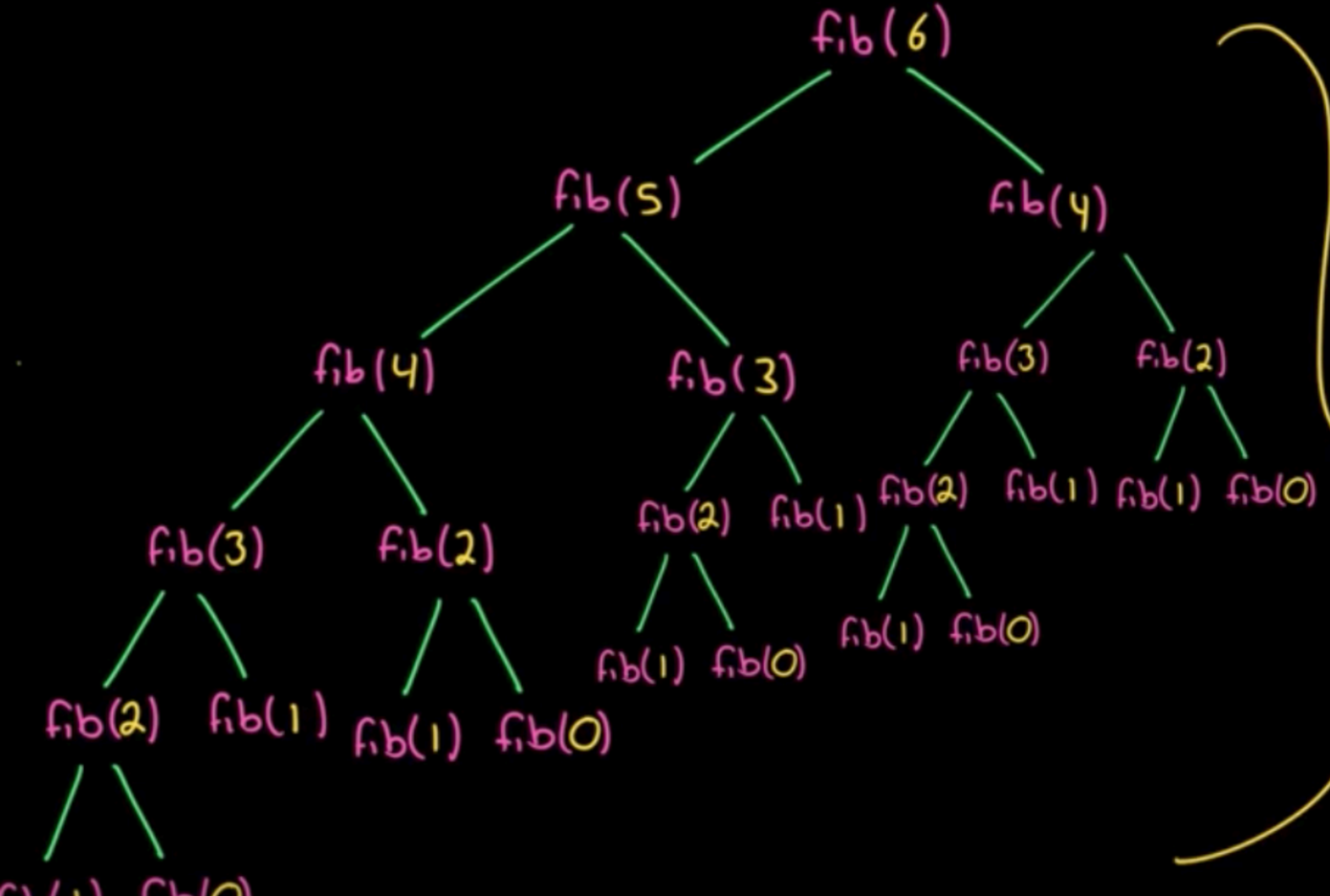
S R

$fib(n) = fib(n-1) + fib(n-2)$

$fib(0) = 0$

$fib(1) = 1$

```
int fib(int n) {
    if (n<=0) {
        return 0;
    } else if (n==1) {
        return 1;
    } else {
        return fib(n-1)+fib(n-2);
    }
}
```



call tree has n levels

Level 1: 1 node

Level 2: 2

Level 3: 4

Level 4: 8

$1 \times 2 \times 2 \times \dots \times 2 = O(2^n)$

N → R

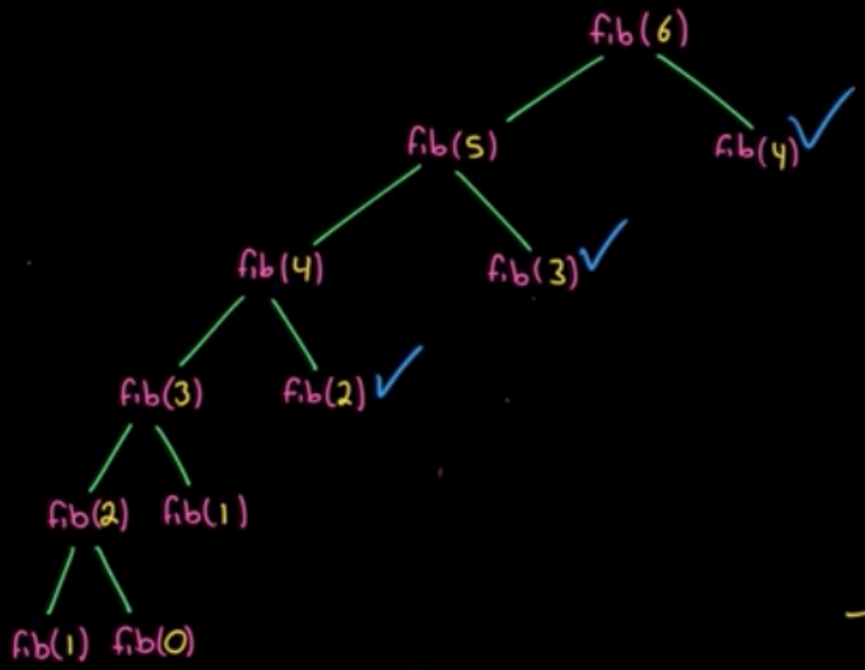
MEMOIZATION + DYNAMIC PROGRAMMING

S R

$fib(n) = fib(n-1) + fib(n-2)$

$fib(0) = 0$

$fib(1) = 1$



memo[2] = 1 ✓

[3] = 2 ✓

[4] = 3 ✓

[5] = 5

```
int fib(int n) {
    if (n<=0) {
        return 0;
    } else if (n==1) {
        return 1;
    } else {
        return fib(n-1)+fib(n-2);
    }
}
```

call tree has n levels

Level 1: 1 node

Level 2: 2

Level 3: 4

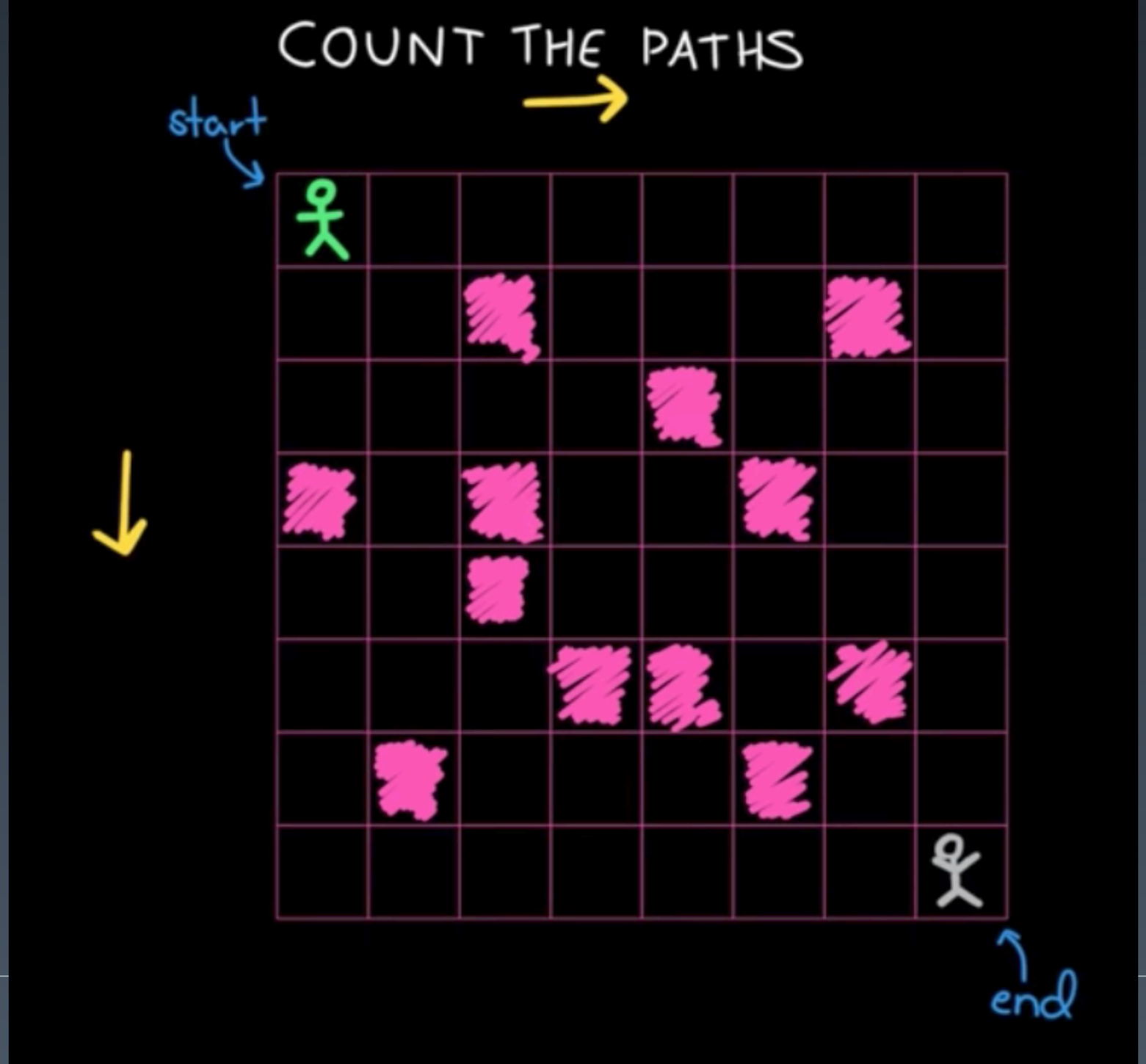
Level 4: 8

$1 \cdot 2 \cdot 2 \cdot \dots \cdot 2 = O(2^n)$

```
int fib(int n, int[] memo) {
    if (n<=0) {
        return 0;
    } else if (n==1) {
        return 1;
    } else if (memo[n]) {
        memo[n]=fib(n-1)+fib(n-2);
    }
    return memo[n];
}
```

MEMOIZATION

↳ O(n) time

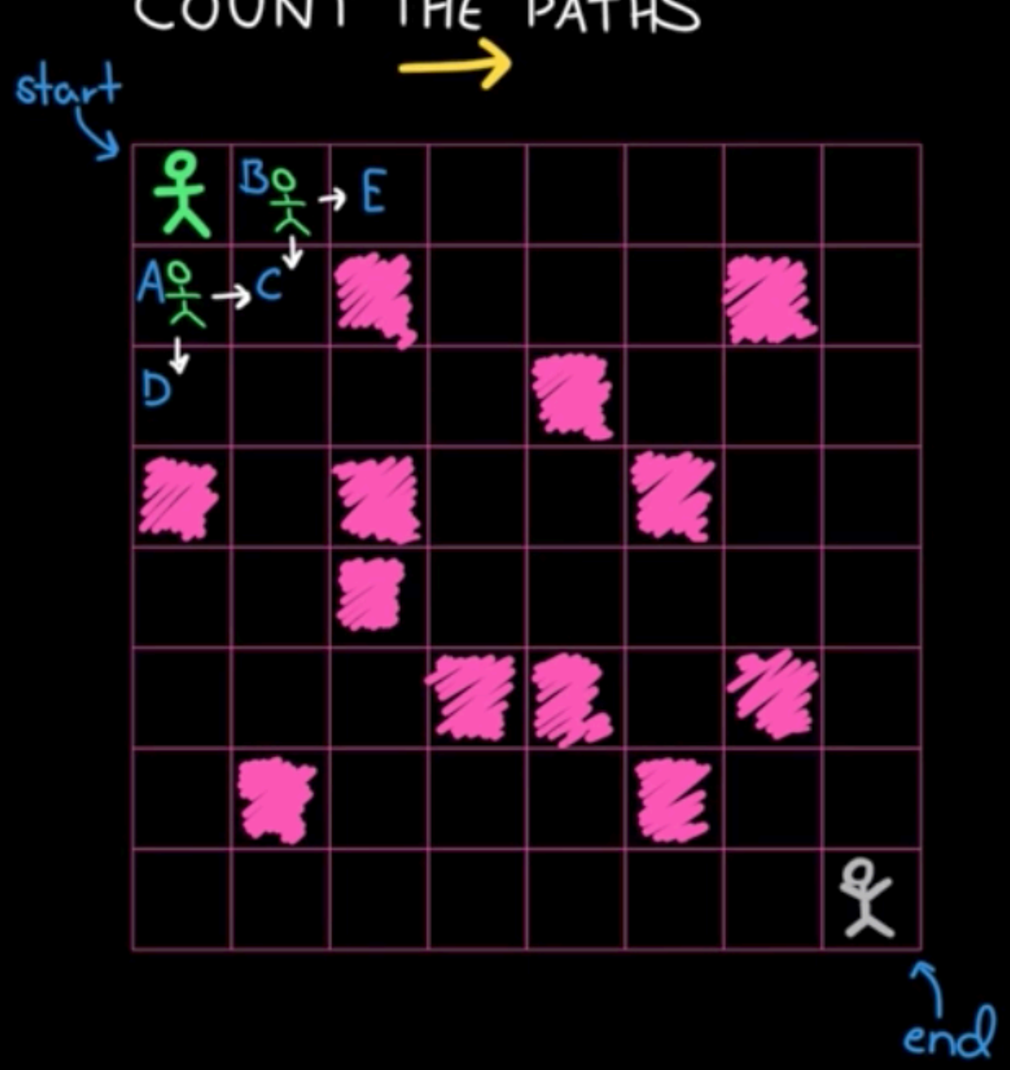


paths(start,end) =

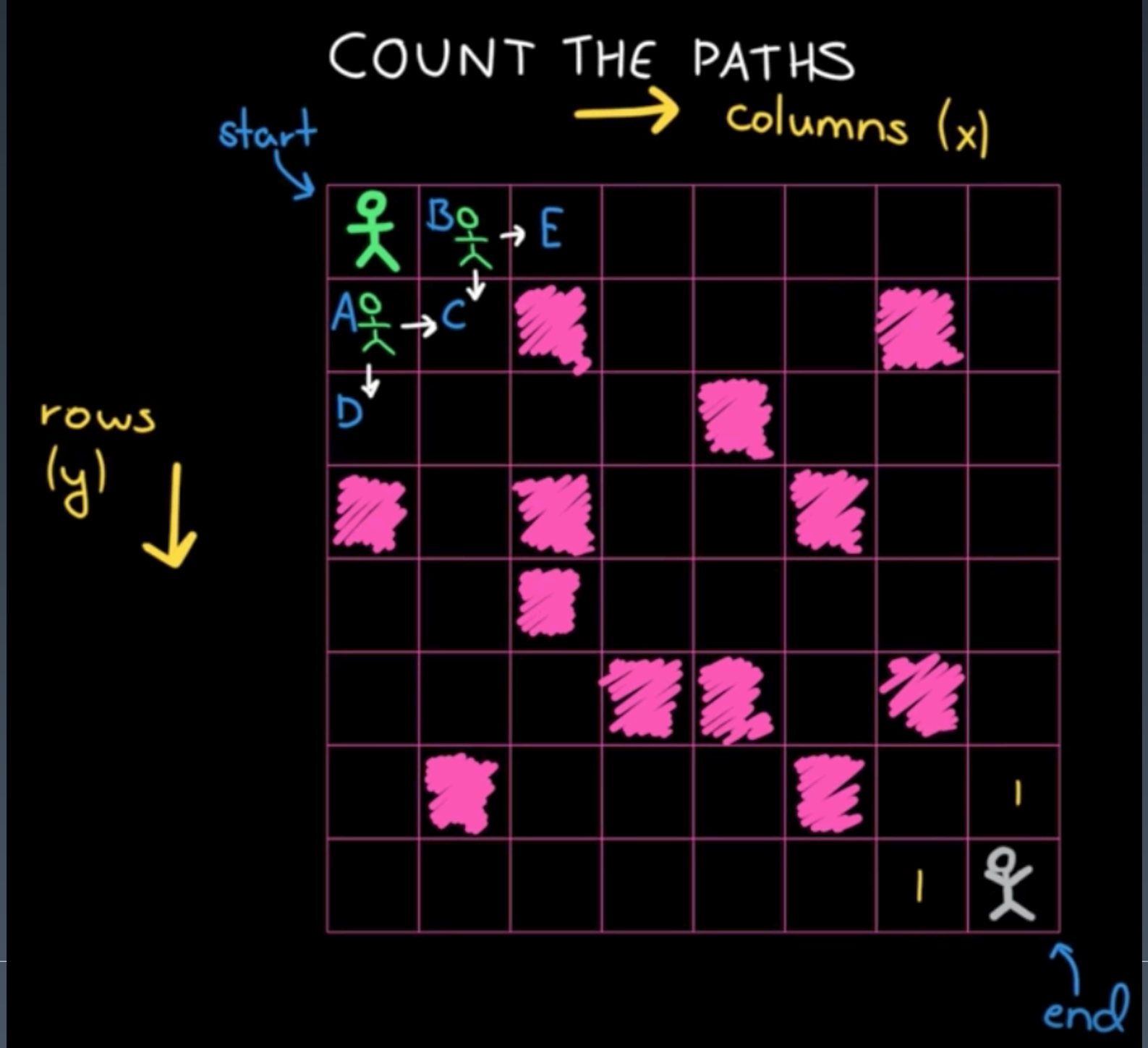
paths(A,end) + paths(B,end)

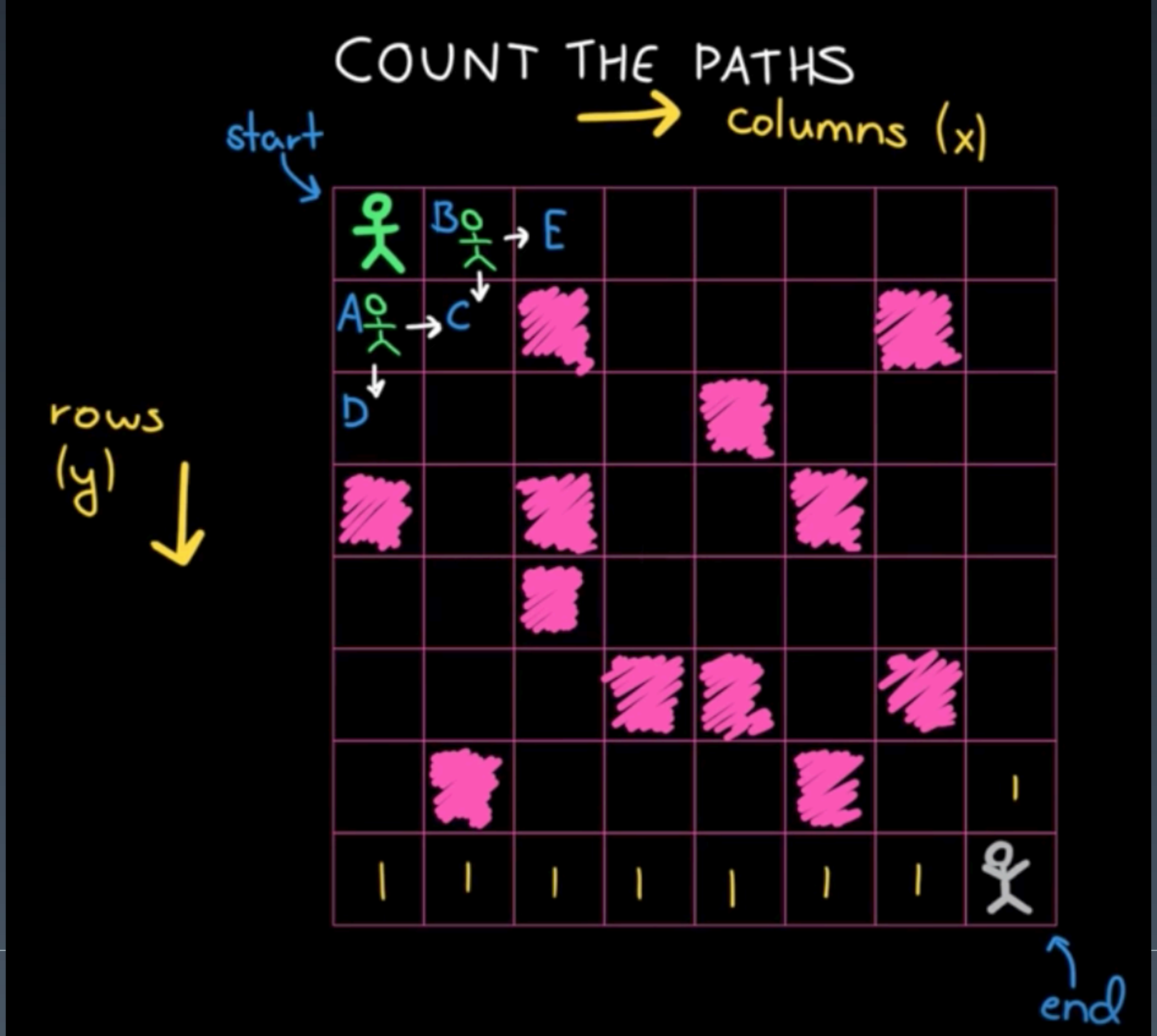
=

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



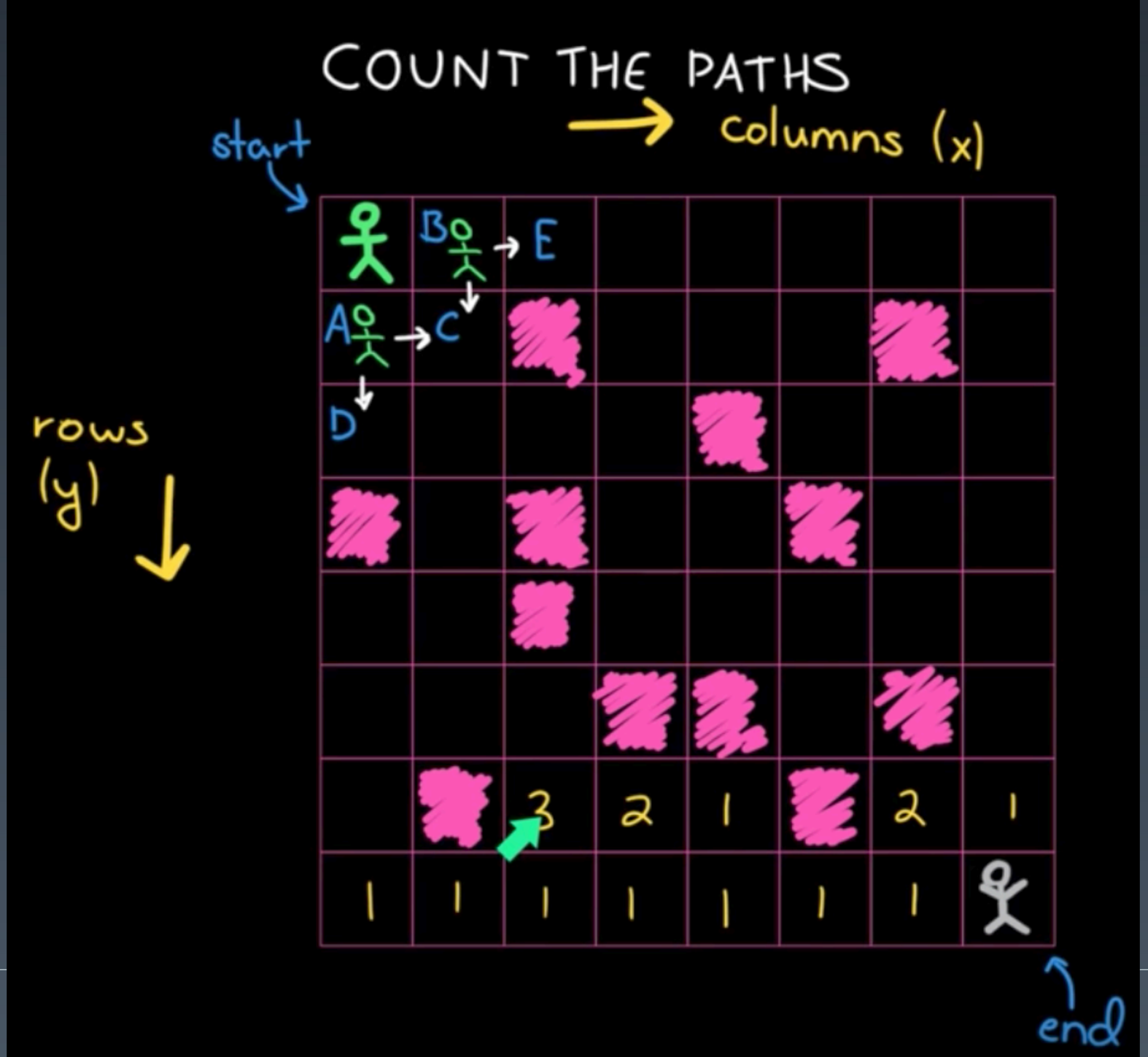

```
int countPaths(boolean[][] grid, int row, int col) {
   if (!validSquare(grid, row, col)) return 0;
   if (isAtEnd(grid, row, col)) return 1;
   return countPaths(grid, row+1, col) + countPaths(grid, row, col+1);
}
```

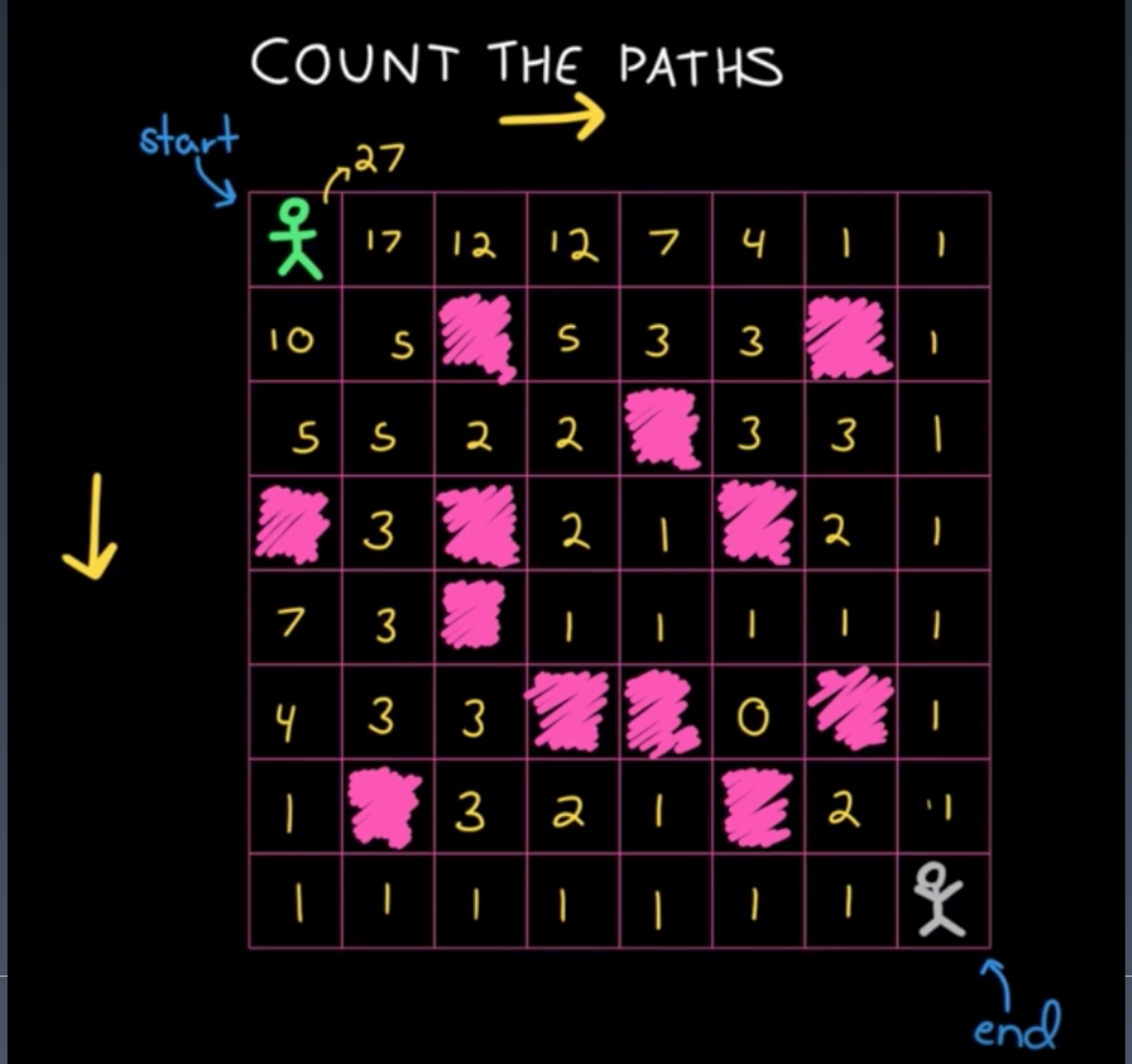

COUNT THE PATHS
columns (x)
start
rows (y)
B
E
A
C
D
1
1
end


COUNT THE PATHS
columns (x)
start
rows (y)
B
E
A
C
D
end

```
    opt[i , j] = opt[i + 1, j] + opt[i, j + 1]
```

```
if a[i, j] = ‘空地’:
    opt[i , j] = opt[i + 1, j] + opt[i, j + 1]
else: // 石头
    opt[i , j] = 0
```


COUNT THE PATHS
columns (x)
start
rows (y)
B
E
A
C
D
3
2
1
2
1
1 1 1 1 1 1 1
end


COUNT THE PATHS
start
27
17 12 12 7 4 1 1
10 5 5 3 3 1
5 5 2 2 3 3 1
3 2 1 2 1
7 3 1 1 1 1 1
4 3 3 0 1
1 3 2 1 2 1
1 1 1 1 1 1 1
end

动态规划 Dynamic Programming

1. 递推！（recursion）
2. 状态的定义 fib[n]
3. 最优子结构 optimized = fib[n]
4. 状态转移方程（DP方程：fib[n] = fib[n-1] + fib[n-2]）

动态规划 Dynamic Programming

1. 打破自己的思维惯性，形成机器思维

2. 理解复杂逻辑的关键

3. 也是职业进阶的要点要领

01 背包问题

假设山洞里共有a,b,c,d ,e这5件宝物（不是5种宝物），它们的重量分别是2,2,6,5,4，它们的价值分别是6,3,5,4,6，现在给你个承重为10的背包, 怎么装背包，可以才能带走最多的财富。

01背包的状态转换方程

f[i,j] = Max{ f[i-1,j-Wi]+Pi(j >= Wi), f[i-1,j] }

f[i,j]表示在前i件物品中选择若干件放在承重为 j 的背包中，可以取得的最大价值。

Pi表示第i件物品的价值。

决策：为了背包中物品总价值最大化，第 i件物品应该放入背包中吗？

预习题目（上课考察）

1. https://leetcode.com/problems/best-time-to-buy-and-sell-stock/#/description

2. https://leetcode.com/problems/climbing-stairs/description/

3. https://leetcode.com/problems/triangle/description/

4. https://leetcode.com/problems/maximum-product-subarray/description/

5. https://leetcode.com/problems/coin-change/description/

预习题目（上课考察）

1. https://leetcode.com/problems/house-robber/
2. https://leetcode.com/problems/house-robber-ii/description/

3. https://leetcode.com/problems/best-time-to-buy-and-sell-stock/#/description
4. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/
5. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/
6. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/
7. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/
8. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/

实战题目

1. https://leetcode-cn.com/problems/perfect-squares/
2. https://leetcode-cn.com/problems/burst-balloons/
3. https://leetcode-cn.com/problems/jump-game/
4. https://leetcode-cn.com/problems/jump-game-ii/
5. https://leetcode-cn.com/problems/unique-paths/
6. https://leetcode-cn.com/problems/unique-paths-ii/
7. https://leetcode-cn.com/problems/unique-paths-iii/
8. https://leetcode-cn.com/problems/coin-change/
9. https://leetcode-cn.com/problems/coin-change-2/
10.

实战题目

1. https://leetcode-cn.com/problems/longest-valid-parentheses/
2. https://leetcode-cn.com/problems/minimum-path-sum/
3. https://leetcode-cn.com/problems/edit-distance/
4. https://leetcode-cn.com/problems/decode-ways
5. https://leetcode-cn.com/problems/maximal-square/
6. https://leetcode-cn.com/problems/max-sum-of-rectangle-no-larger-than-k/
7. https://leetcode-cn.com/problems/frog-jump/
8. https://leetcode.com/problems/split-array-largest-sum
9. https://leetcode-cn.com/problems/student-attendance-record-ii/
10. https://leetcode-cn.com/problems/task-scheduler/
11. https://leetcode-cn.com/problems/palindromic-substrings/
12. https://leetcode-cn.com/problems/minimum-window-substring/

THANKS!