动态规划（Dynamic Programming)

本节内容

1. 递归+记忆化 —> 递推
2. 状态的定义：opt[n], dp[n], fib[n]
3. 状态转移方程：opt[n] = best_of(opt[n-1], opt[n-2], ...)
4. 最优子结构

斐波那契数列

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

递推公式：F[n] = F[n-1] + F[n-2]

N → R

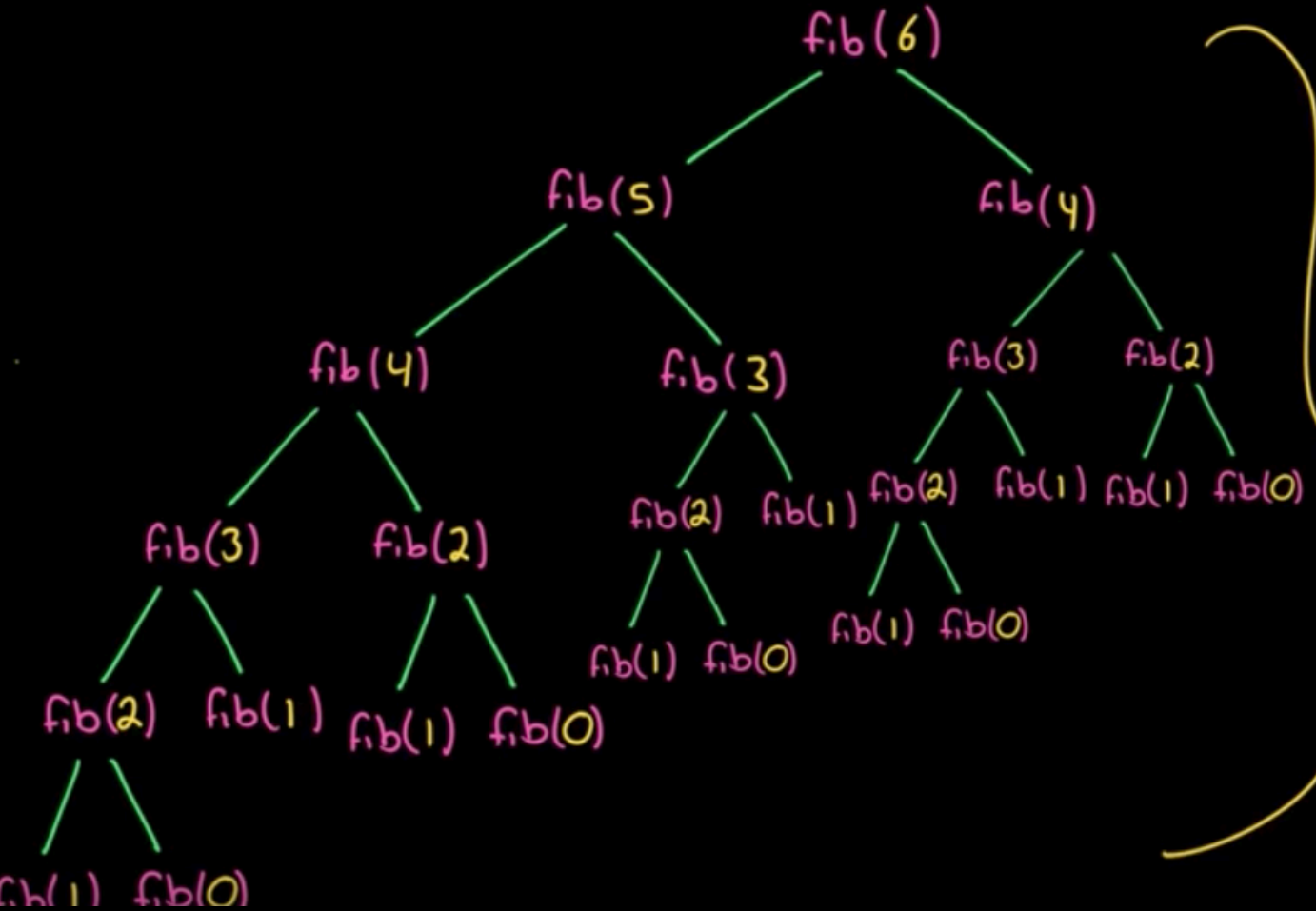
MEMOIZATION + DYNAMIC PROGRAMMING

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

$fib(0) = 0$

$fib(1) = 1$

```
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

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

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

N → R

E

O MEMOIZATION + C

I DYNAMIC PROGRAMMING V

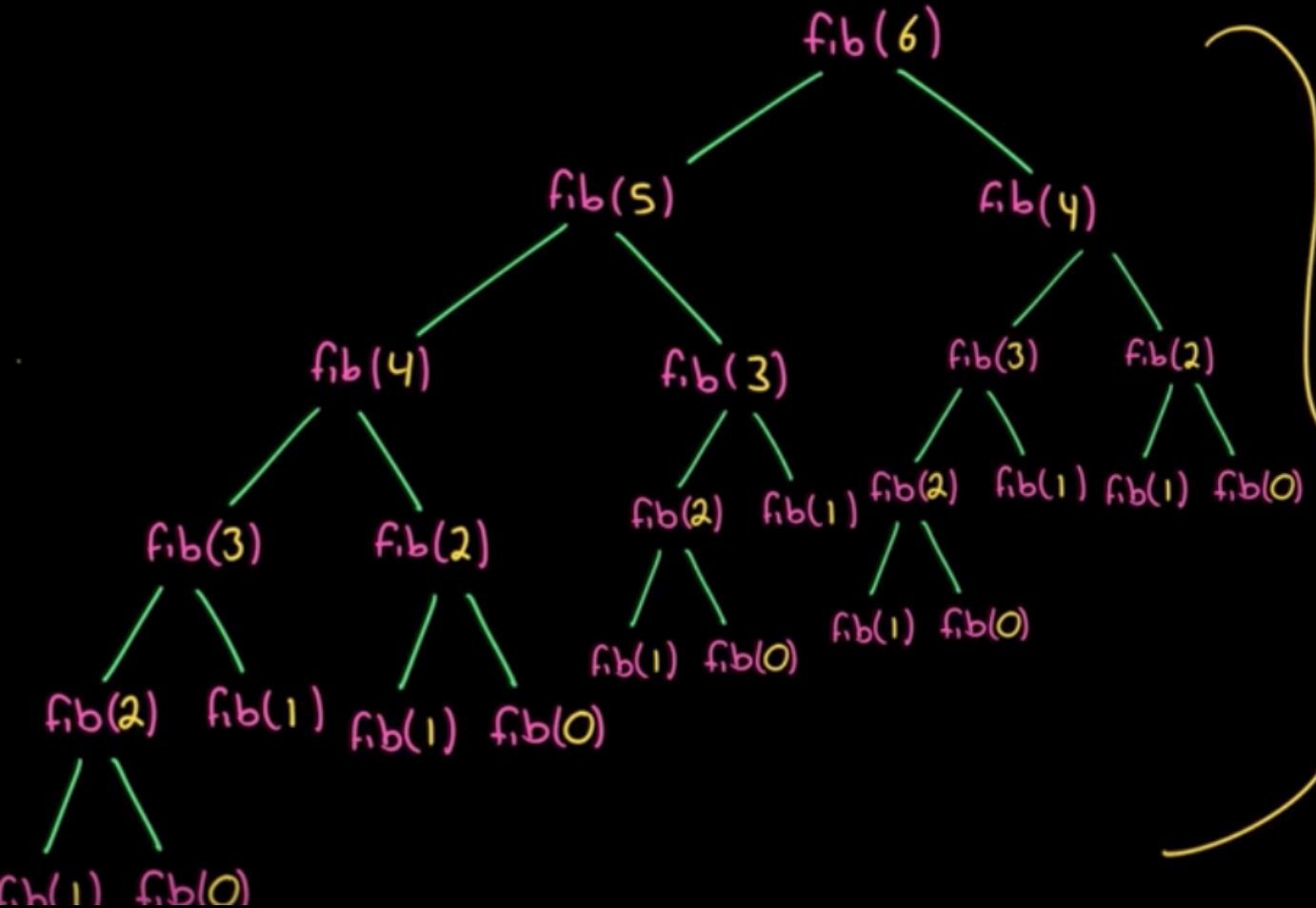
S R

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

$fib(0) = 0$

$fib(1) = 1$

```
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

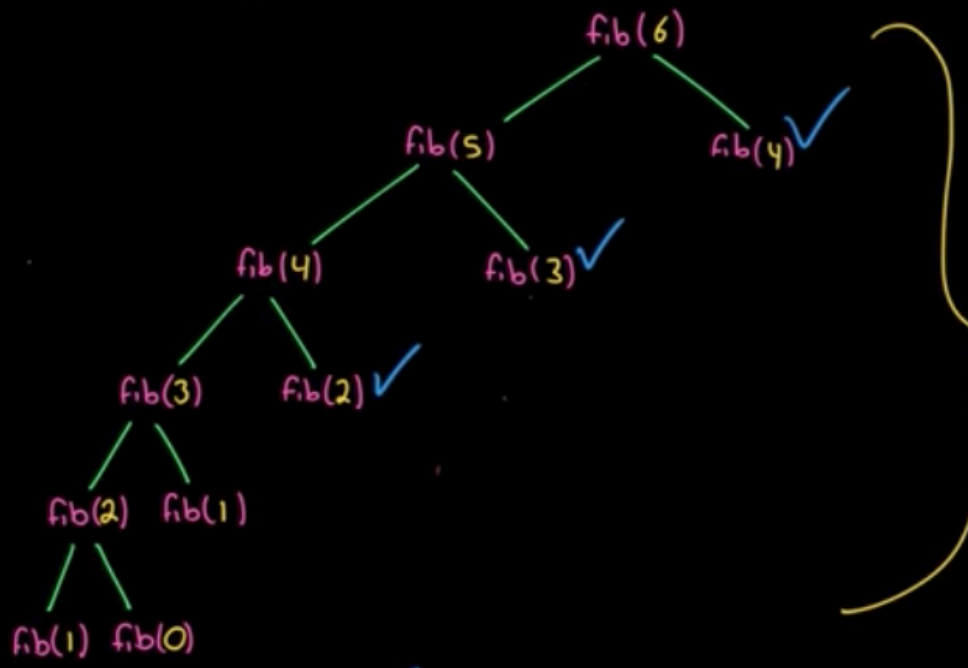
$1 \times 2 \times 2 \times \dots \times 2 = O(2^n)$

N → R
E
O MEMOIZATION C
+
I 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];
}
```

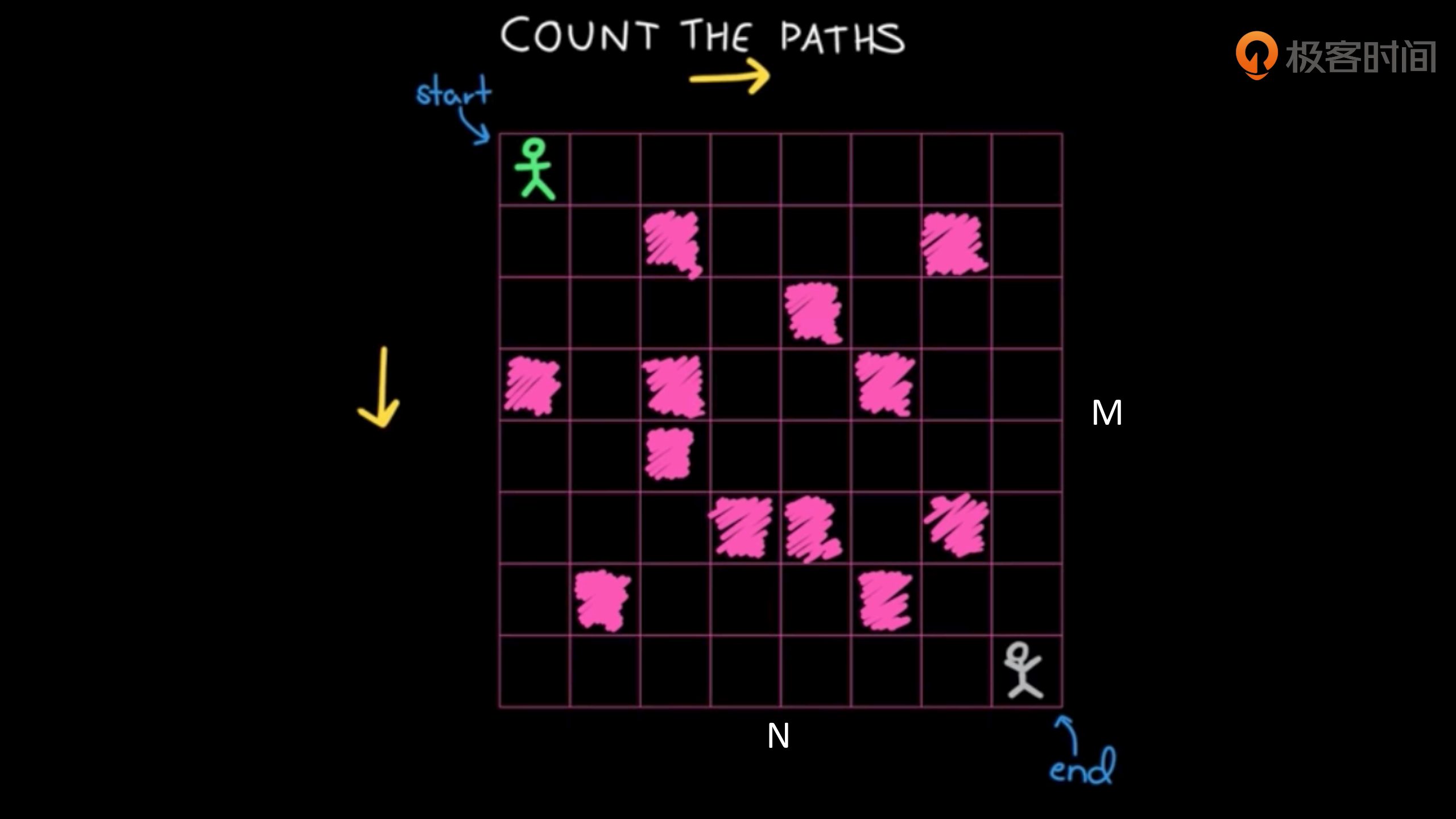
MEMOIZATION

↳ O(n) time

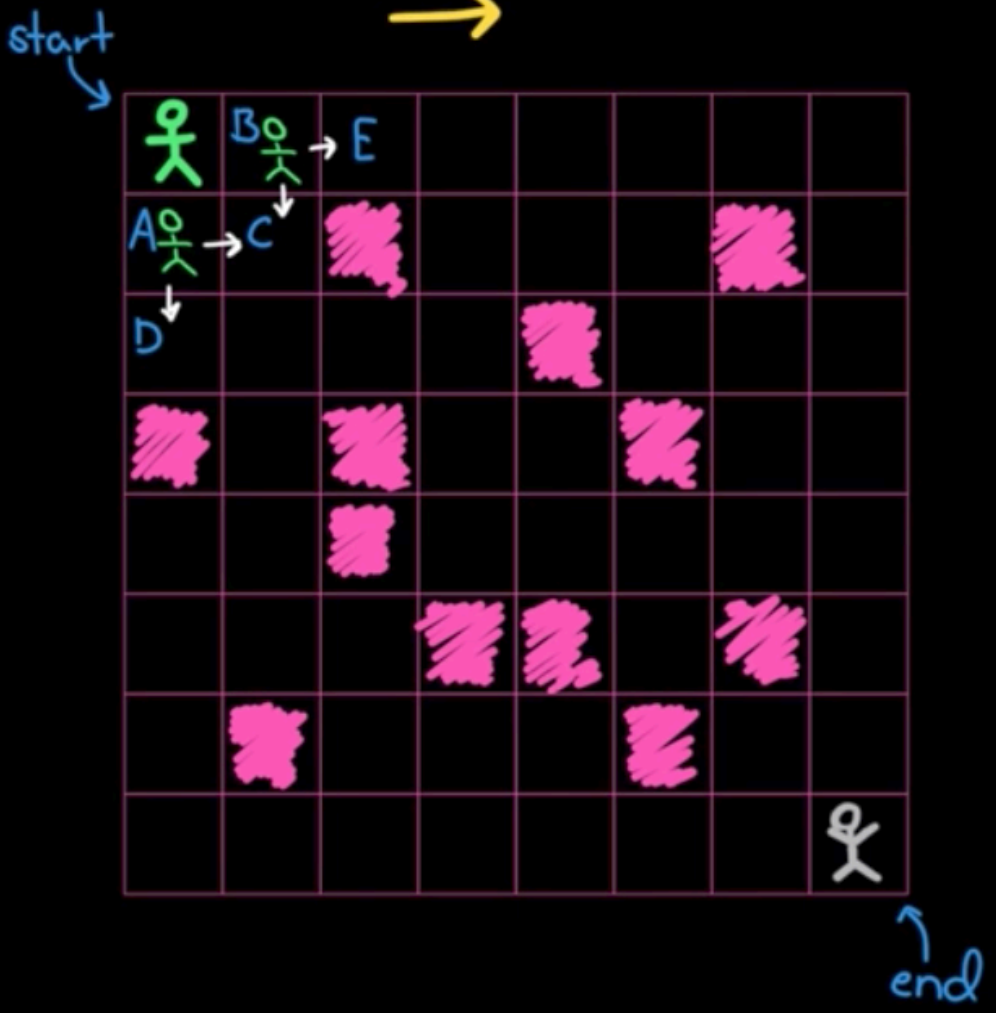
递归 + 记忆化 ==> 递推

递推公式：F[n] = F[n-1] + F[n-2]

```
F[0] = 0, F[1] = 1;
for (int i = 2; i <= n; ++i) {
    F[i] = F[i - 1] + F[i - 2];
}
```


COUNT THE PATHS
start
M
N
end

COUNT THE PATHS



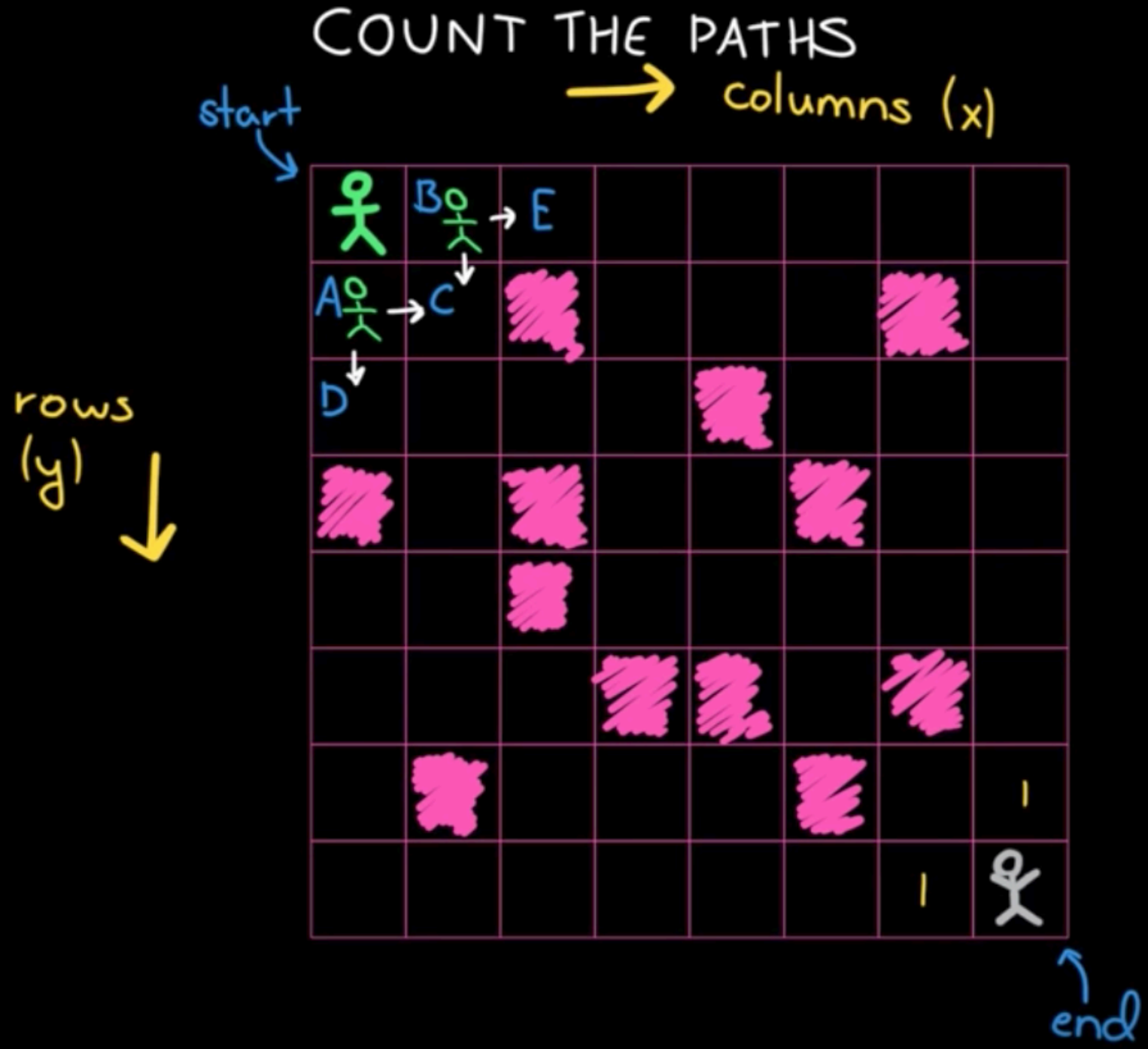
paths(start,end) =

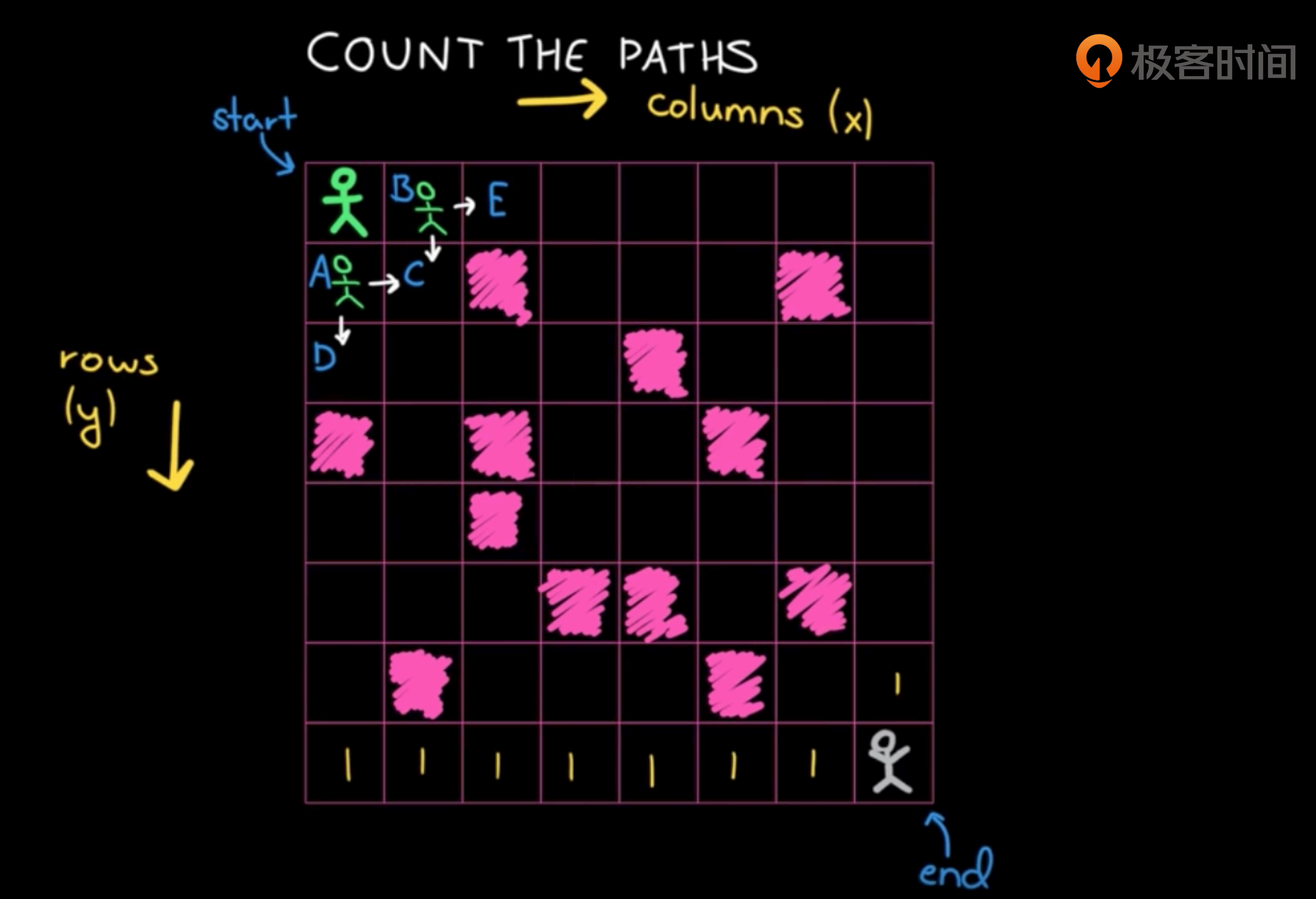
paths(A,end) + paths(B,end)

paths(A,end) = paths(D,end) + paths(C,end)

paths(B,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);
}
```

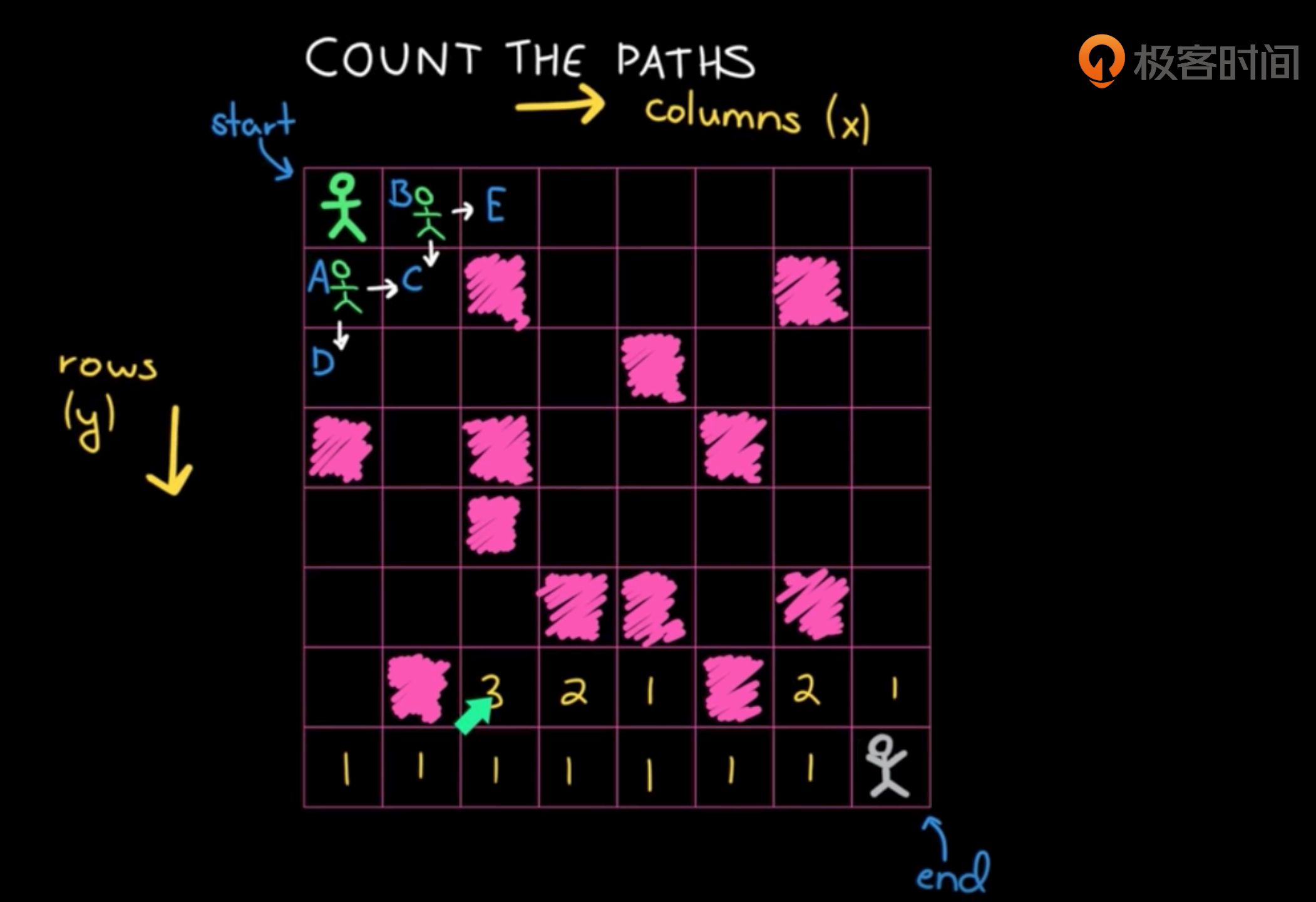

COUNT THE PATHS
columns (x)
start
rows (y)
B
E
A
C
D
1
1
end


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

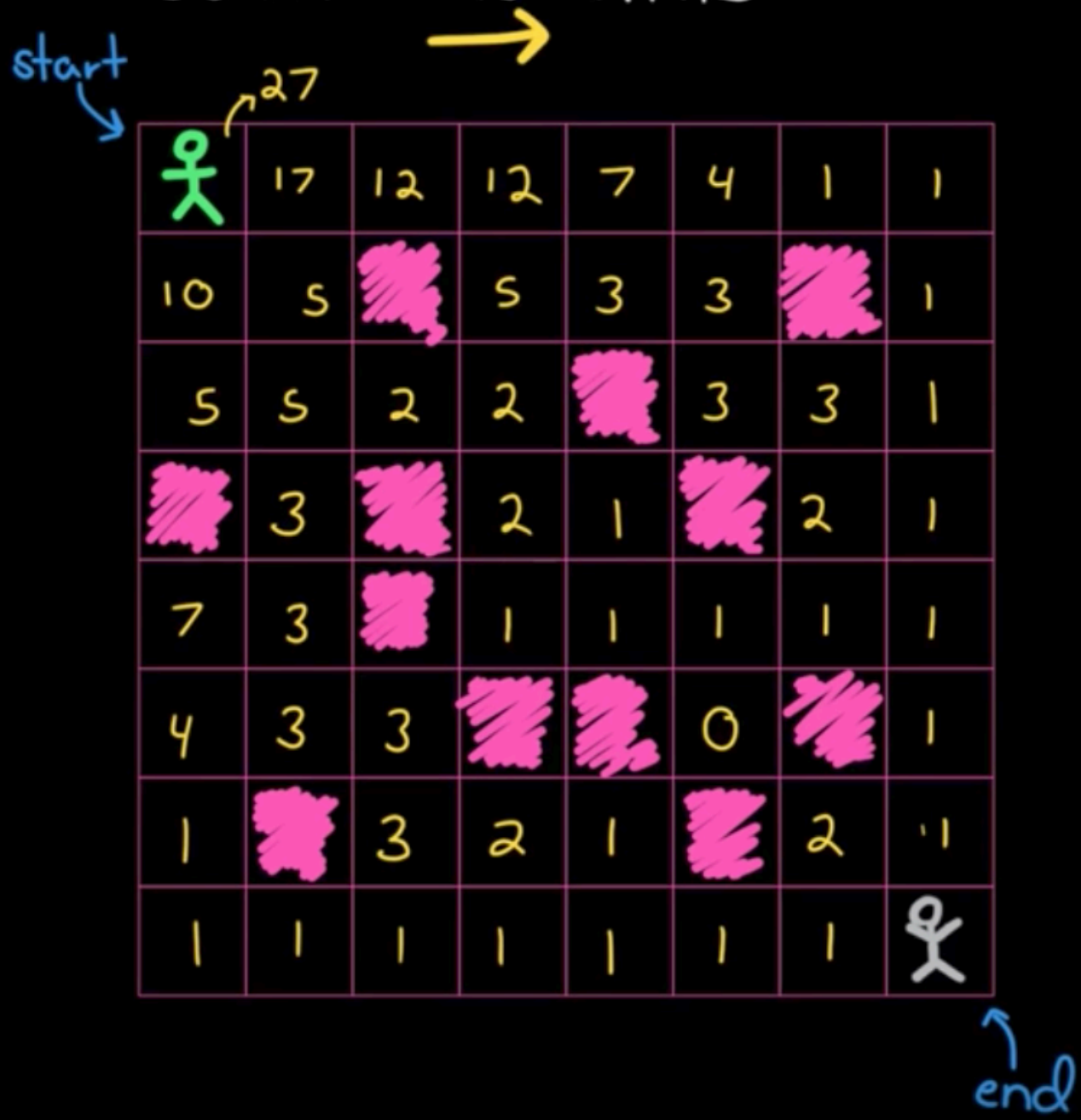
```
opt[i , j] = opt[i − 1, j] + opt[i, j − 1]

_____________________________________

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



动态规划 Dynamic Programming

1. 递推！ （递推 + 记忆化）
2. 状态的定义：opt[n], dp[n], fib[n]
3. 状态转移方程：opt[n] = best_of(opt[n-1], opt[n-2], …)
4. 最优子结构

动态规划 vs 回溯 vs 贪心算法

- 回溯（递归）— 重复计算
- 贪心算法 — 永远局部最优
- 动态规划 — 记录局部最优子结构 / 多种记录值

实战题目

1. https://leetcode.com/problems/climbing-stairs/description/
2. https://leetcode.com/problems/triangle/description/
3. https://leetcode.com/problems/maximum-product-subarray/description/
4. https://leetcode.com/problems/best-time-to-buy-and-sell-stock/#/description
5. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/
6. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/
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-cooldown/
9. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/
10. https://leetcode.com/problems/longest-increasing-subsequence
11. https://leetcode.com/problems/coin-change/
12. https://leetcode.com/problems/edit-distance/