遞迴 Recursive

x

動態規劃 Dynamic Programming

Yu-Hsuan Chen

2019 Spring, Introduction to Programming

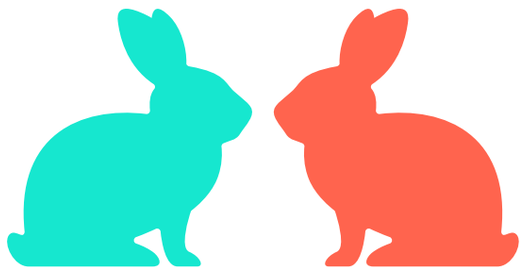
遞迴？

從數學的角度看：數列的下一項由前一項或多項計算而來

從電腦的角度看：將大問題拆解成相同的小問題處理(Divide and Conquer)

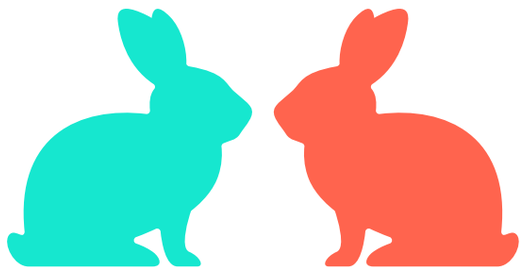
Fibonacci Sequence

Fibonacci Sequence



- 兔子的故事

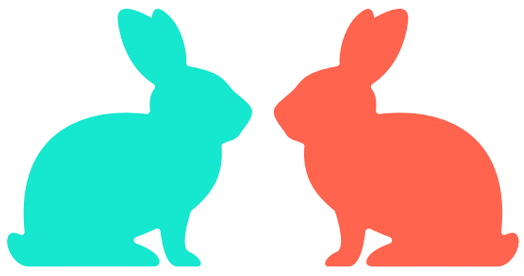
Fibonacci Sequence



- 兔子的故事

- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1

Fibonacci Sequence

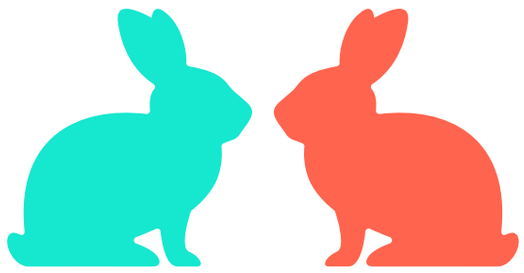


- 兔子的故事

- `F(n) = F(n-1) + F(n-2)`
  `F(0) = 1, F(1) = 1`

- 逐個推導

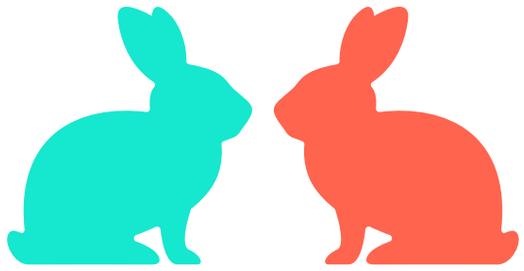
Fibonacci Sequence

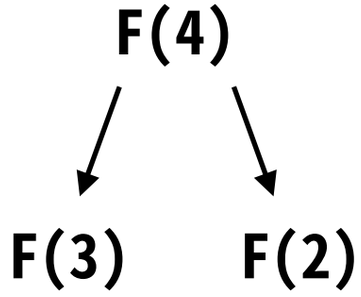


F(4)

- 兔子的故事
- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1
- 逐個推導

Fibonacci Sequence



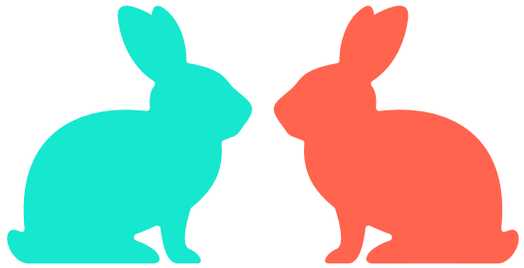


- 兔子的故事

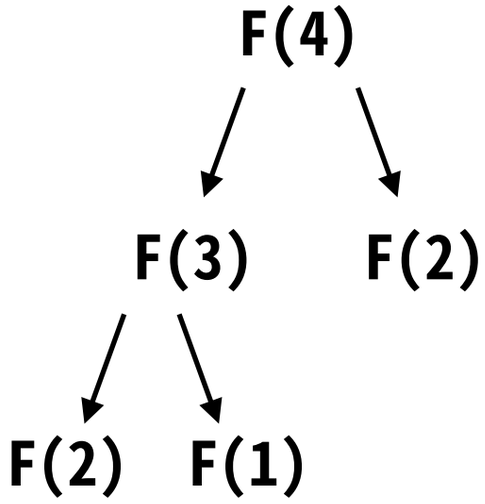
- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1

- 逐個推導

Fibonacci Sequence



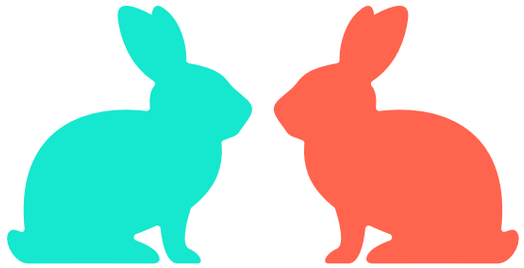
- 兔子的故事



- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1

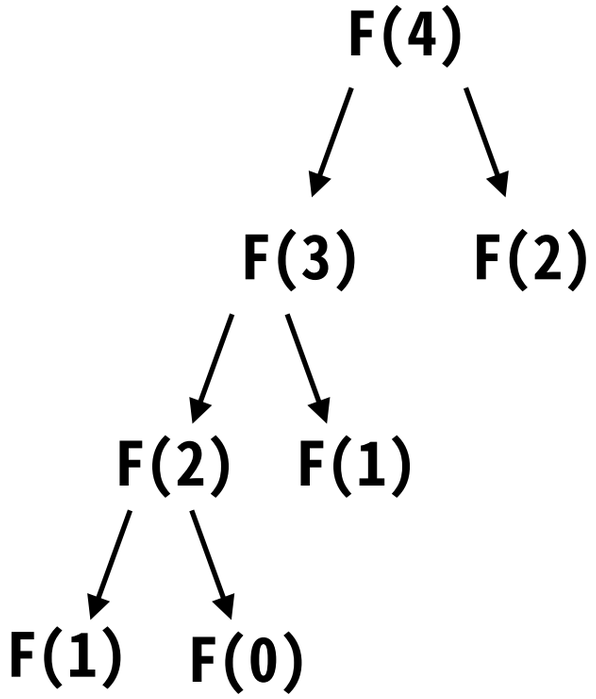
- 逐個推導

Fibonacci Sequence



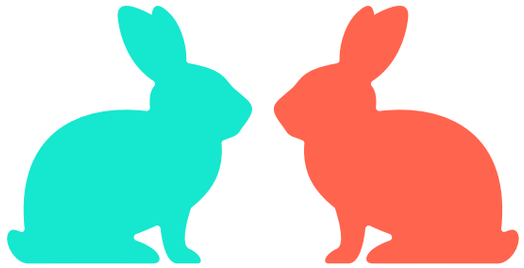
- 兔子的故事

- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1



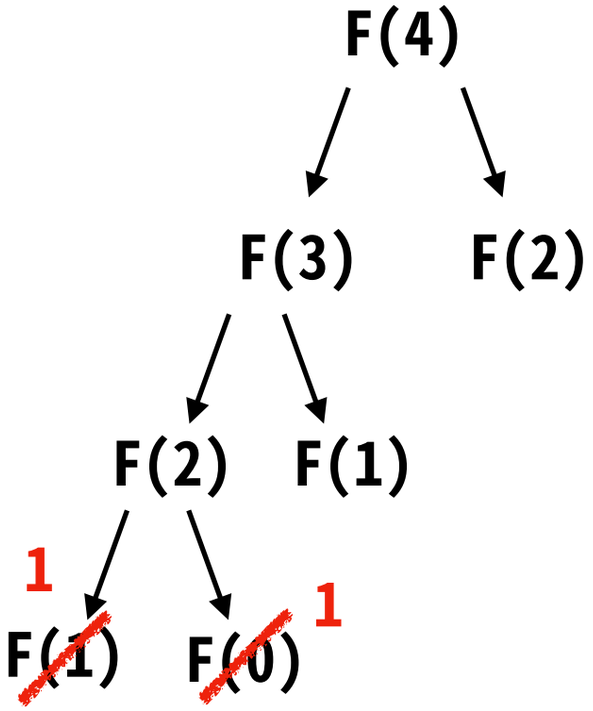
- 逐個推導

Fibonacci Sequence



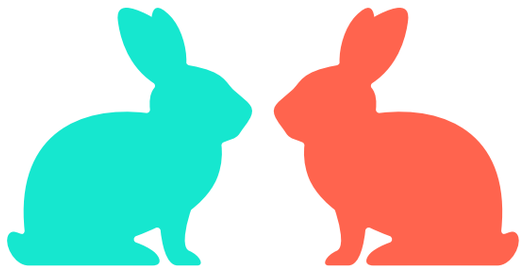
- 兔子的故事

- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1

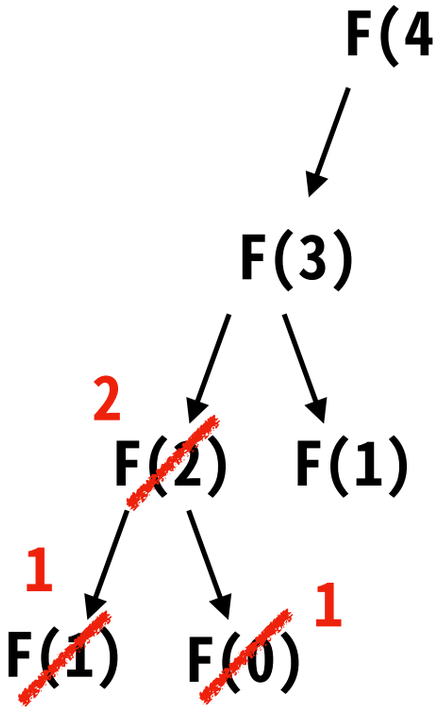


- 逐個推導

Fibonacci Sequence



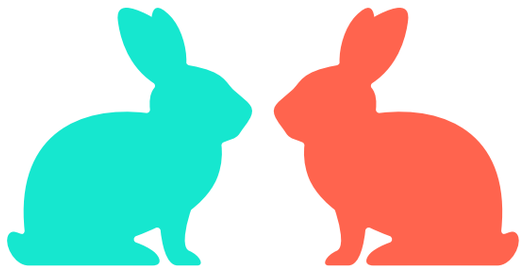
- 兔子的故事



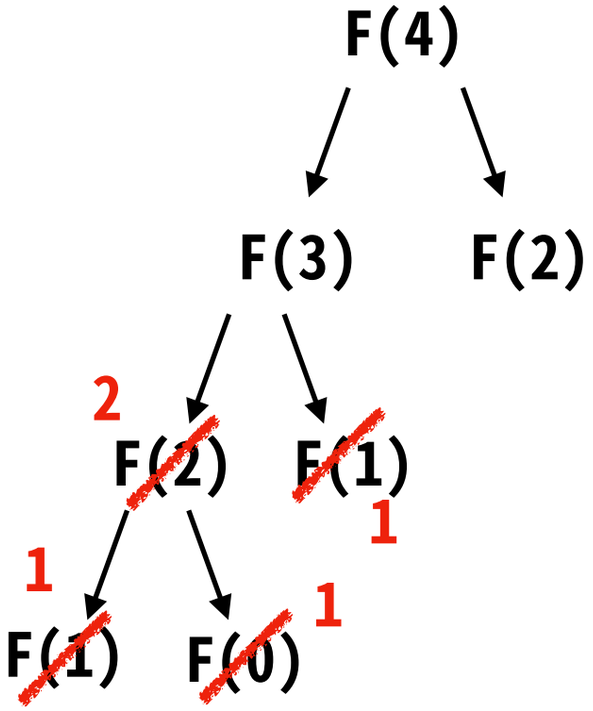
- `F(n) = F(n-1) + F(n-2)`
  `F(0) = 1, F(1) = 1`

- 逐個推導

Fibonacci Sequence



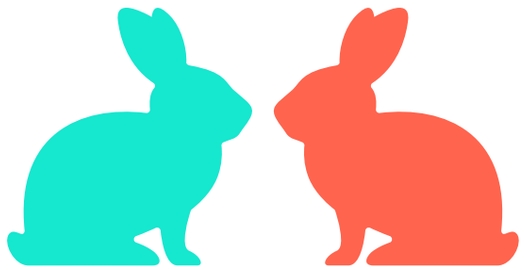
- 兔子的故事



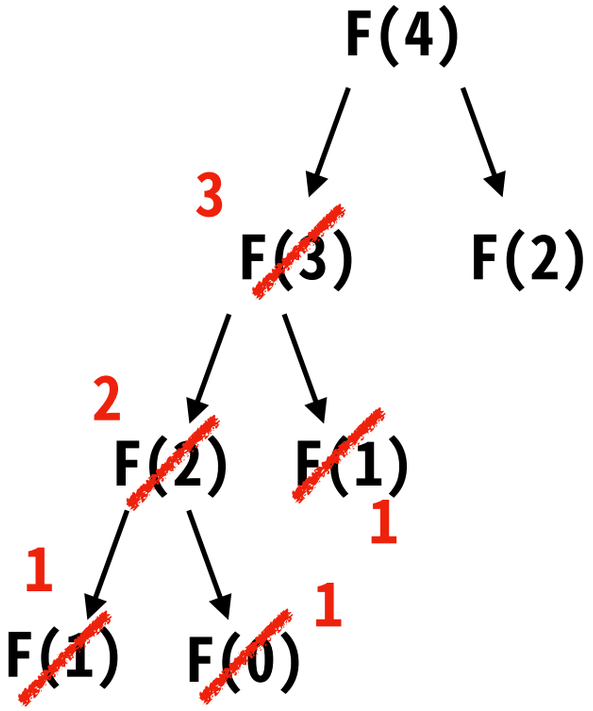
- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1

- 逐個推導

Fibonacci Sequence



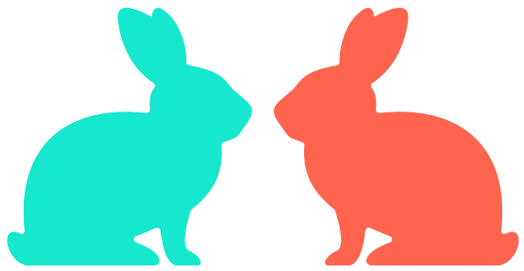
- 兔子的故事



- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1

- 逐個推導

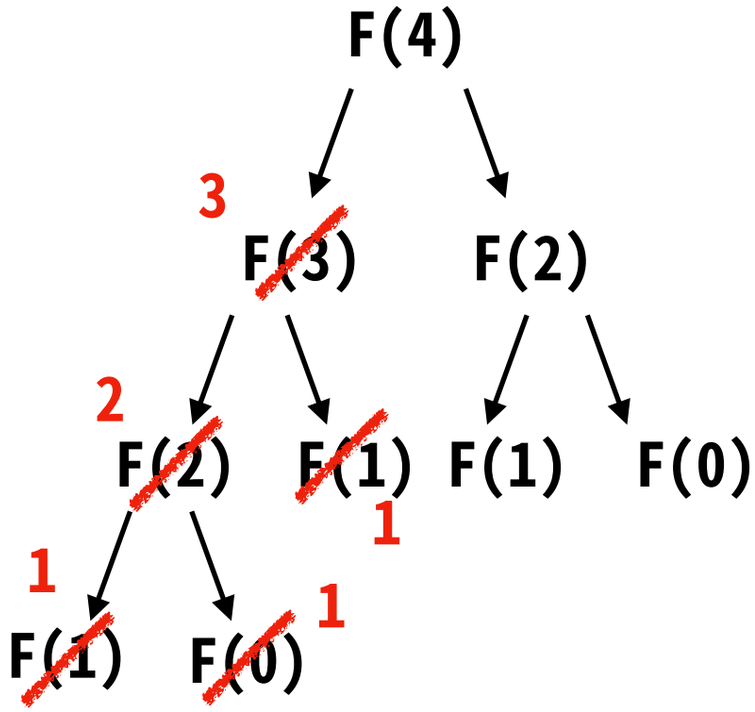
Fibonacci Sequence



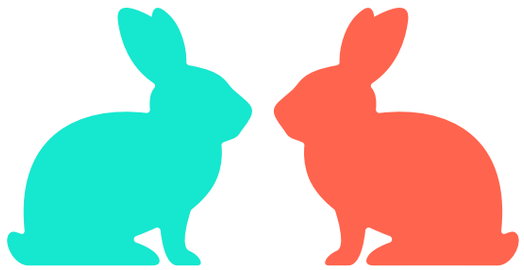
- 兔子的故事

- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1

- 逐個推導



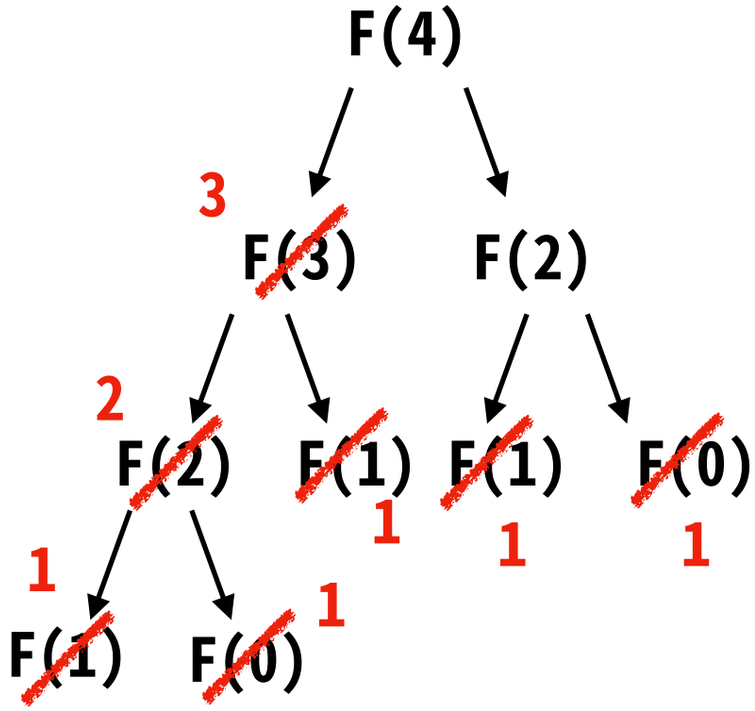
Fibonacci Sequence



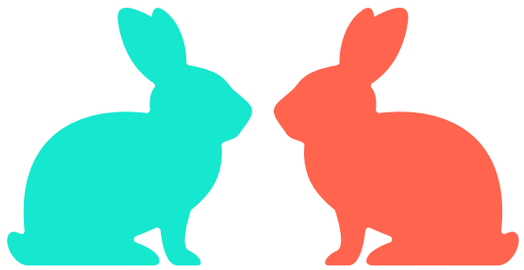
- 兔子的故事

- `F(n) = F(n-1) + F(n-2)`
  `F(0) = 1, F(1) = 1`

- 逐個推導



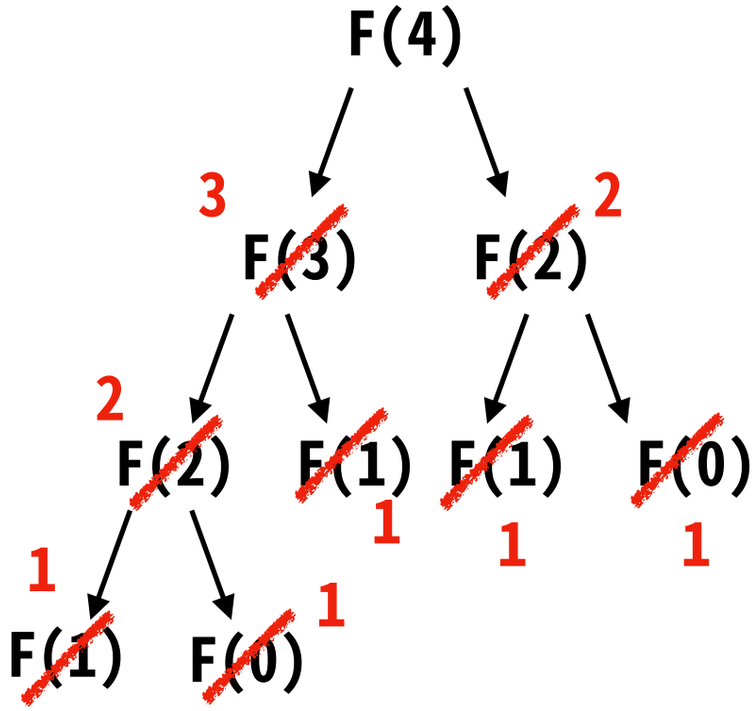
Fibonacci Sequence



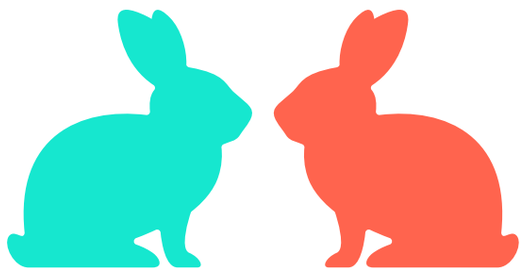
- 兔子的故事

- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1

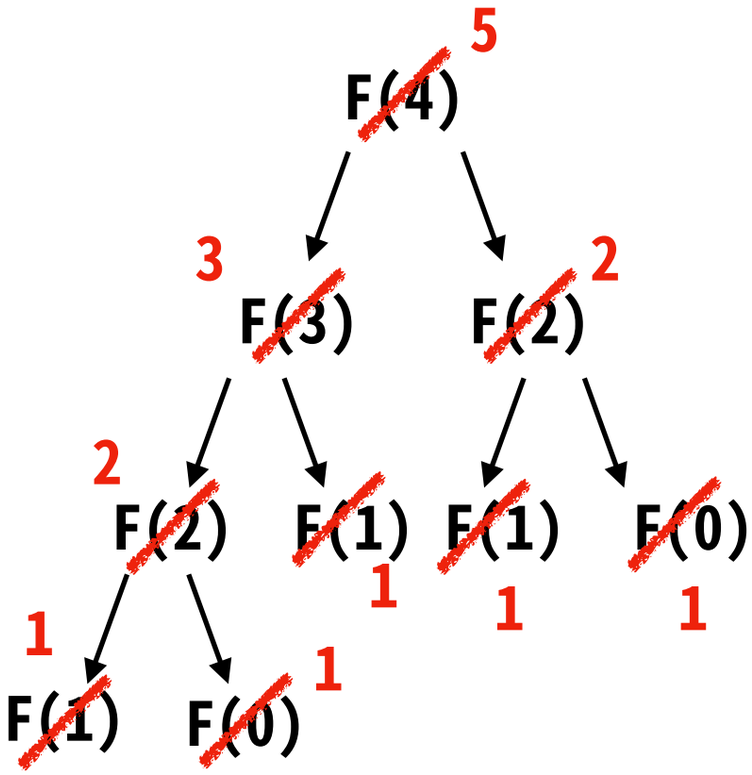
- 逐個推導



Fibonacci Sequence



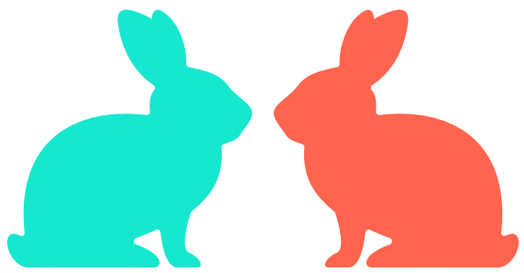
- 兔子的故事



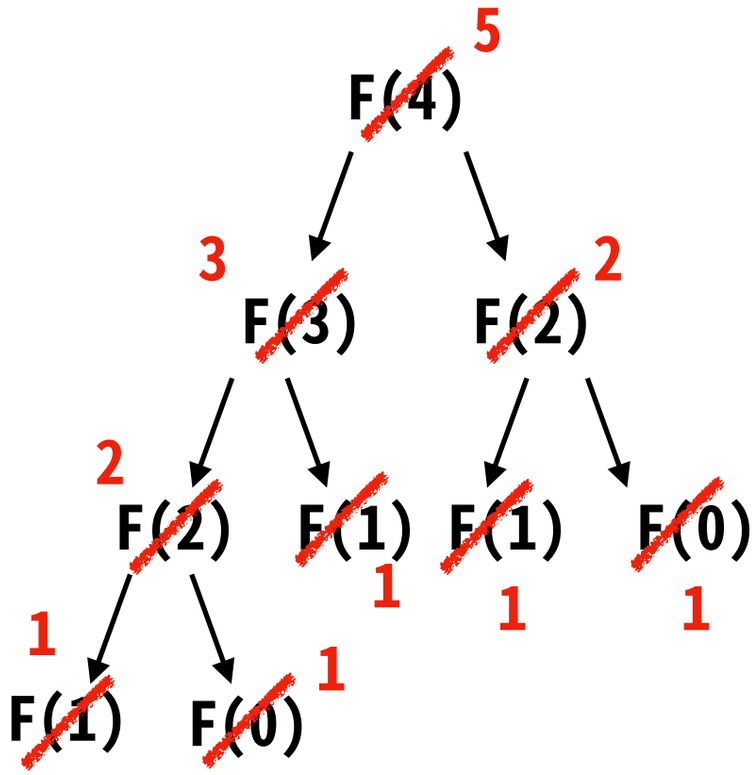
- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1

- 逐個推導

Fibonacci Sequence



- 兔子的故事



- F(n) = F(n-1) + F(n-2)
  F(0) = 1, F(1) = 1

- 逐個推導

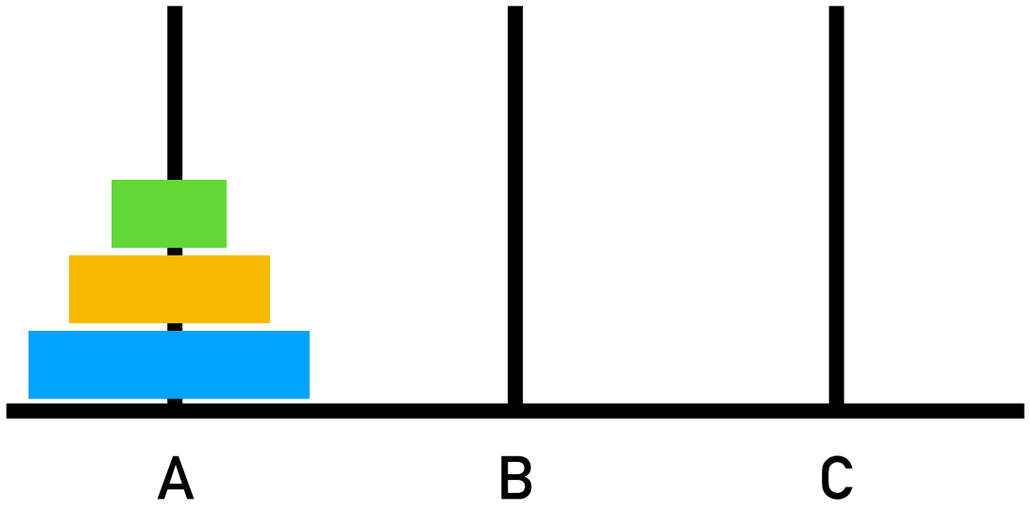
```
int fib(int n)
{
    if(n<2)
        return 1;
    else return f(n-1)+f(n-2);
}
```

Tower of Hanoi

- 三根柱子(peg)、數個大小不一的盤子(disk)
- 一次移動一個盤子
- 大的不可以壓在小的上面

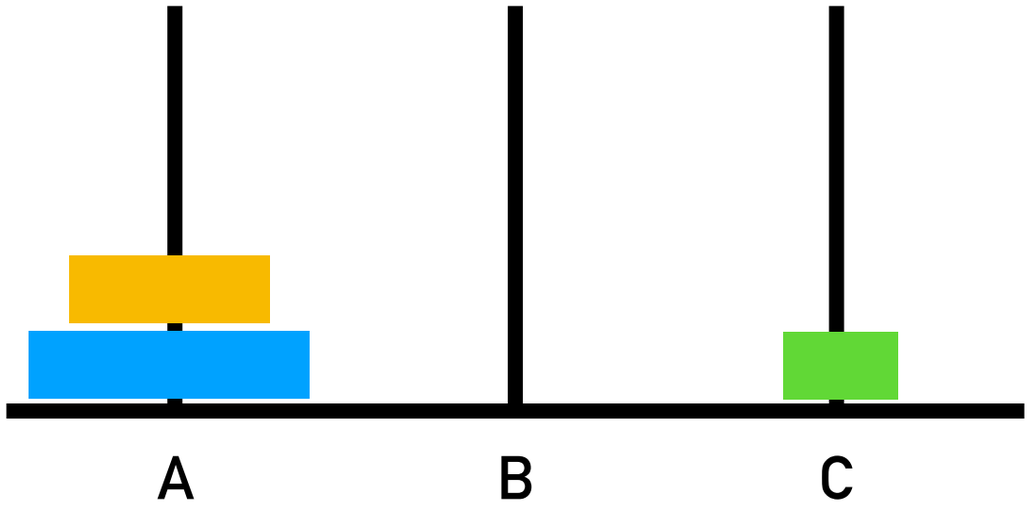
Tower of Hanoi

- 三根柱子(peg)、數個大小不一的盤子(disk)
- 一次移動一個盤子
- 大的不可以壓在小的上面



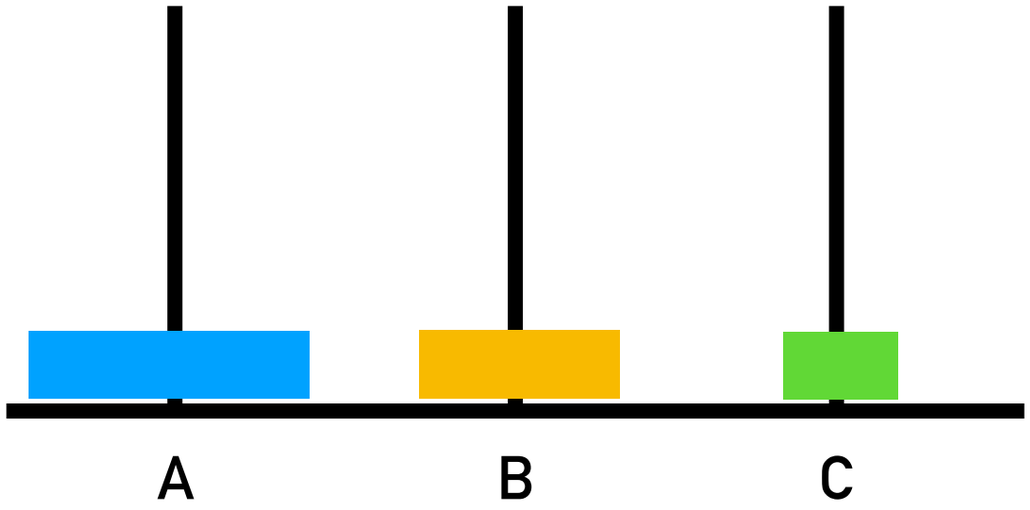
Tower of Hanoi

- 三根柱子(peg)、數個大小不一的盤子(disk)
- 一次移動一個盤子
- 大的不可以壓在小的上面



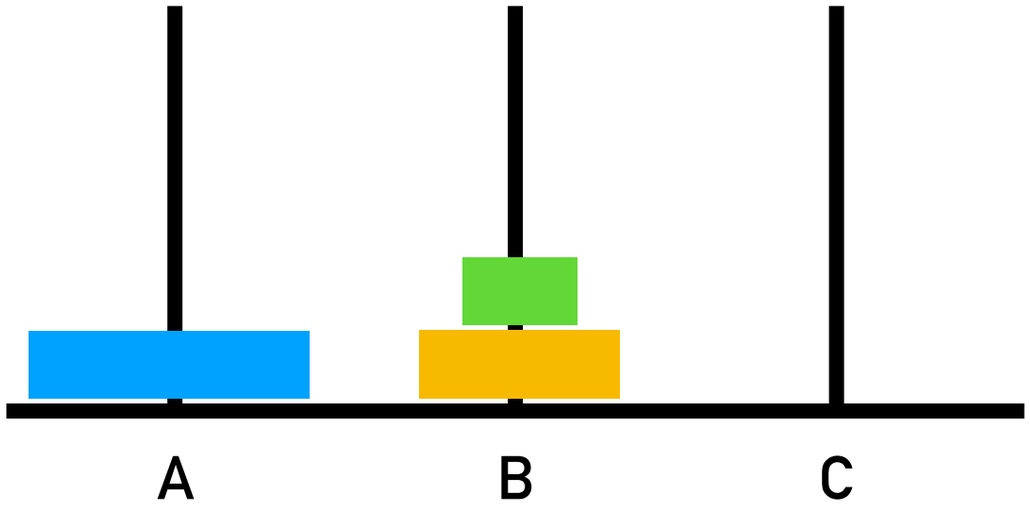
Tower of Hanoi

- 三根柱子(peg)、數個大小不一的盤子(disk)
- 一次移動一個盤子
- 大的不可以壓在小的上面



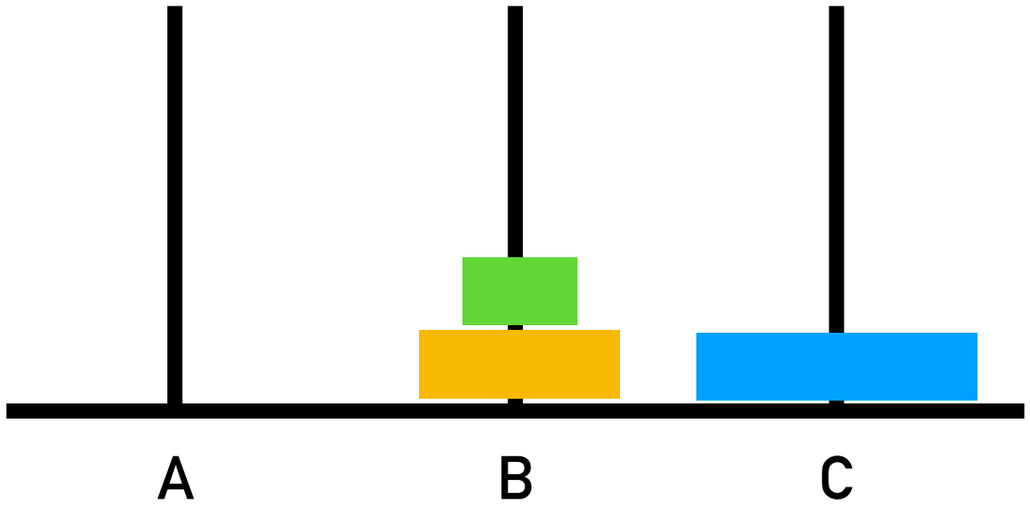
Tower of Hanoi

- 三根柱子(peg)、數個大小不一的盤子(disk)
- 一次移動一個盤子
- 大的不可以壓在小的上面



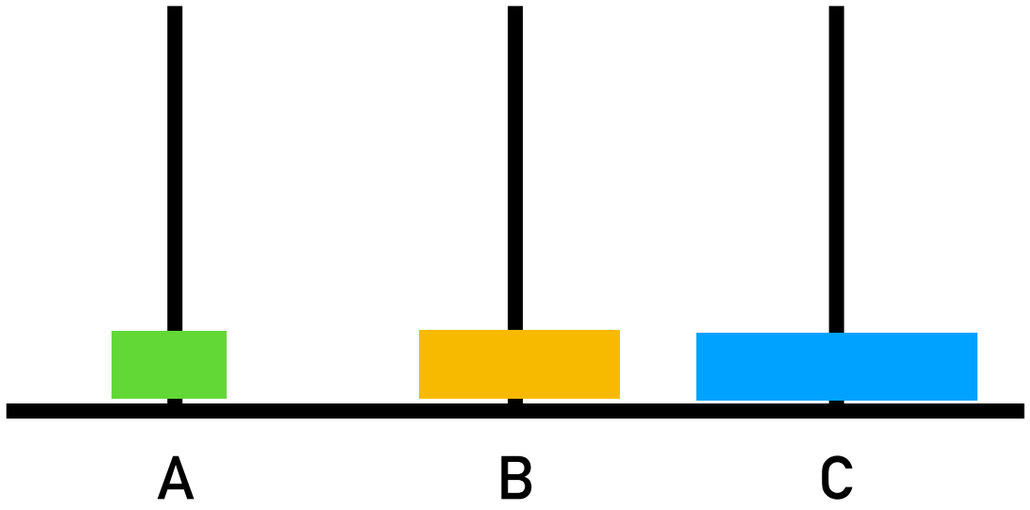
Tower of Hanoi

- 三根柱子(peg)、數個大小不一的盤子(disk)
- 一次移動一個盤子
- 大的不可以壓在小的上面



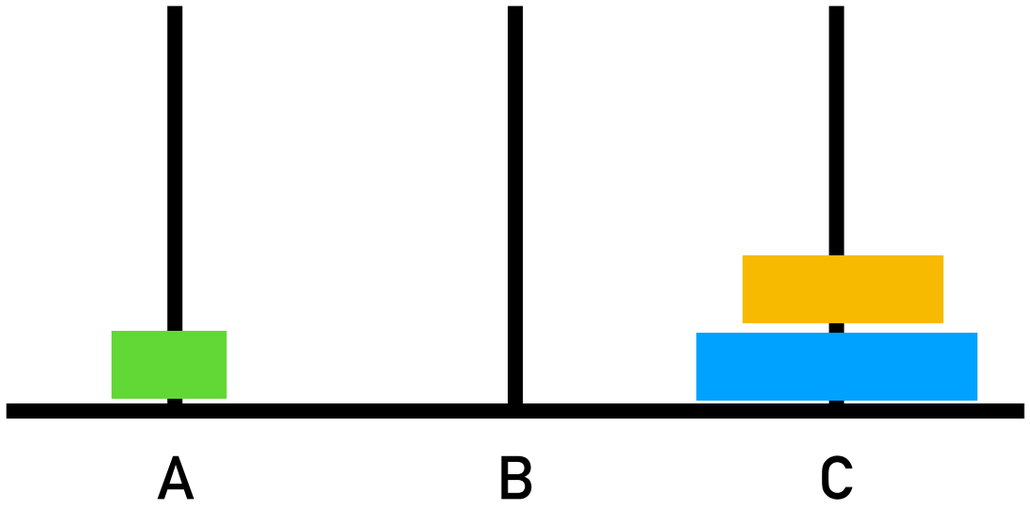
Tower of Hanoi

- 三根柱子(peg)、數個大小不一的盤子(disk)
- 一次移動一個盤子
- 大的不可以壓在小的上面



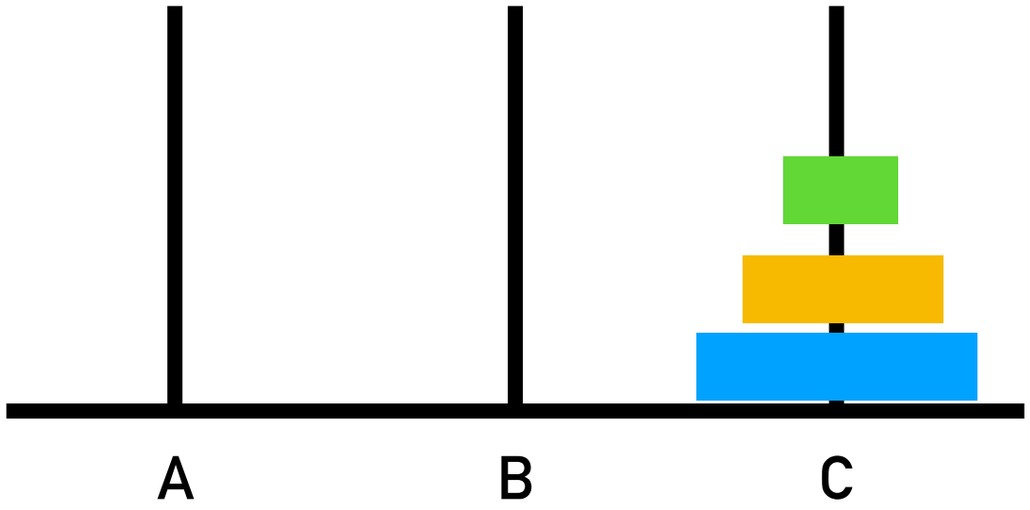
Tower of Hanoi

- 三根柱子(peg)、數個大小不一的盤子(disk)
- 一次移動一個盤子
- 大的不可以壓在小的上面



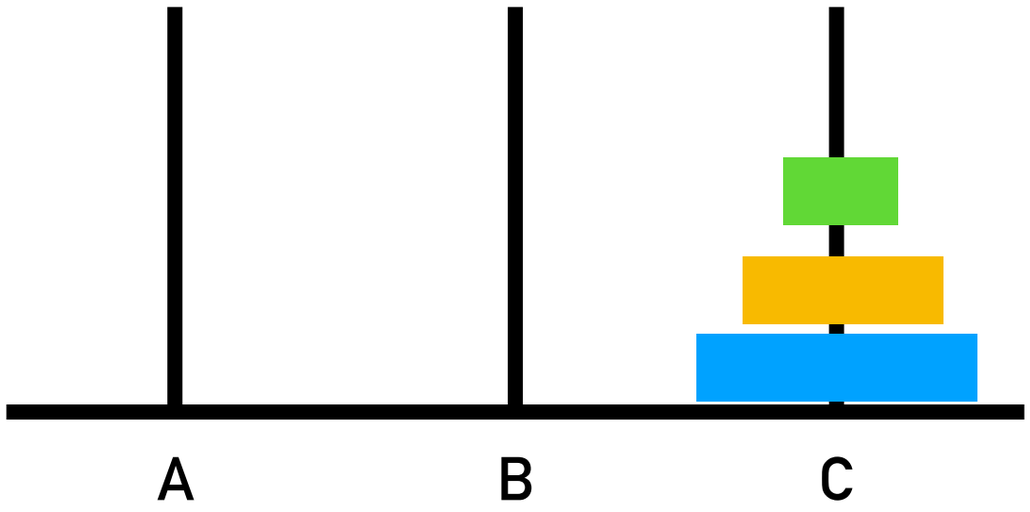
Tower of Hanoi

- 三根柱子(peg)、數個大小不一的盤子(disk)
- 一次移動一個盤子
- 大的不可以壓在小的上面



Tower of Hanoi

- 三根柱子(peg)、數個大小不一的盤子(disk)
- 一次移動一個盤子
- 大的不可以壓在小的上面



搬動N個盤子=
先搬N-1個盤子到B柱
再把最大的盤子移動到C柱
最後把N-1個盤子移回C柱

```
void hanoi(int n,char A,char B,char C)
{
    if(n==1)
        print(“Move” n “from” A “to” C);
    hanoi(n-1, A, C, B);
    hanoi(1, A, B, C);
    hanoi(n-1, B, A, C);
}
```

參數意義：A是來源柱、C是目標柱、B是輔助柱

Depth-First Search

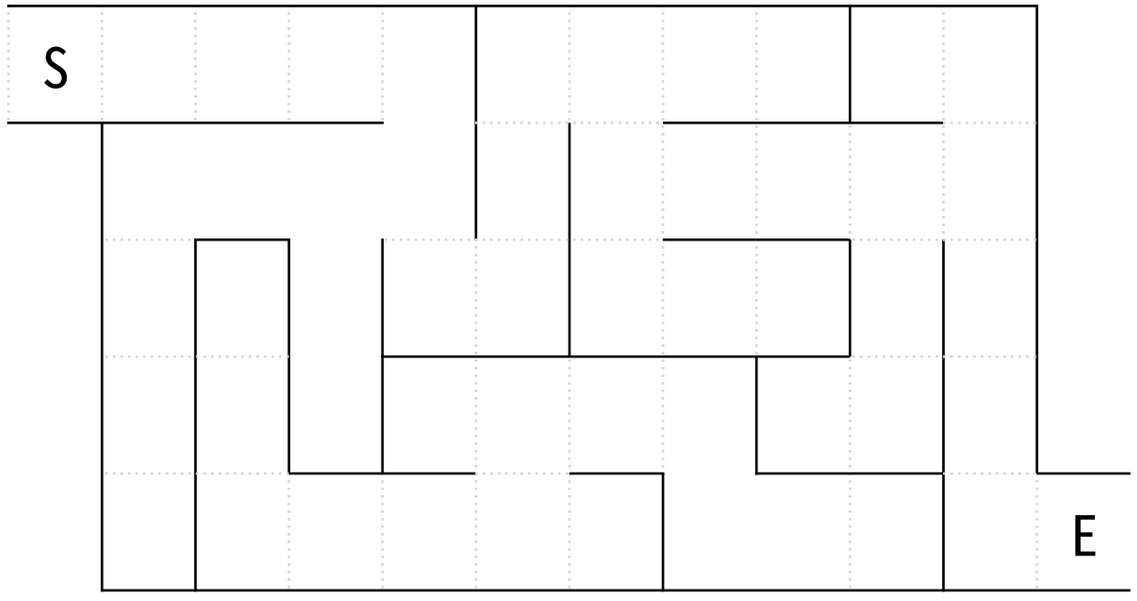
深度優先搜尋

- Graph跟Tree皆有此概念
- 先碰到的點就先處理
- 應用：走迷宮、Graph Theory

Depth-First Search

深度優先搜尋

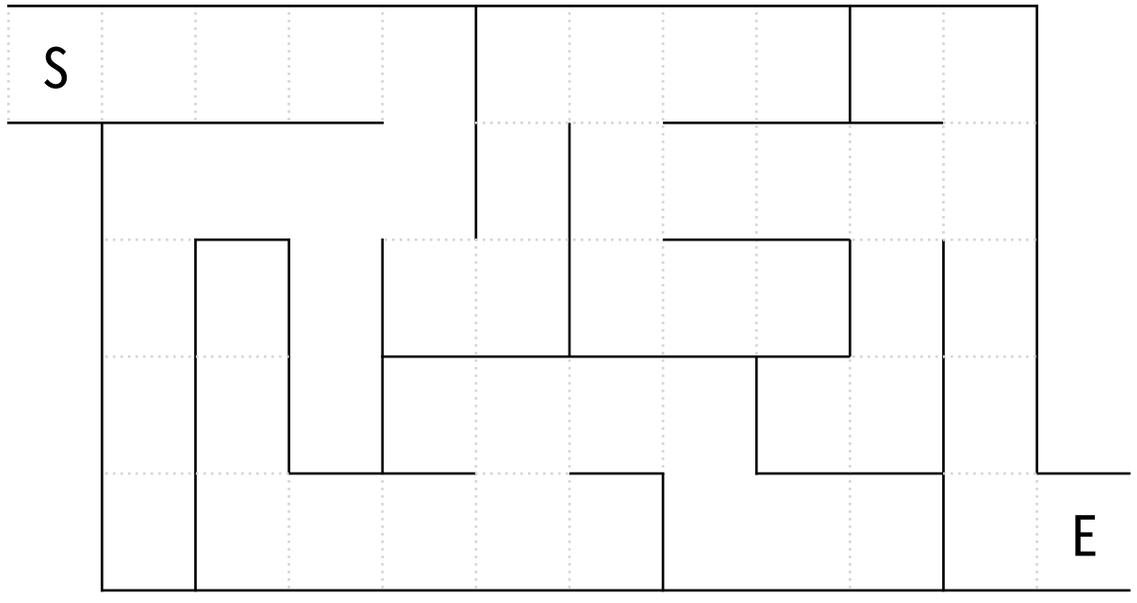
- Graph跟Tree皆有此概念
- 先碰到的點就先處理
- 應用：走迷宮、Graph Theory



Maze

Depth-First Search

深度優先搜尋

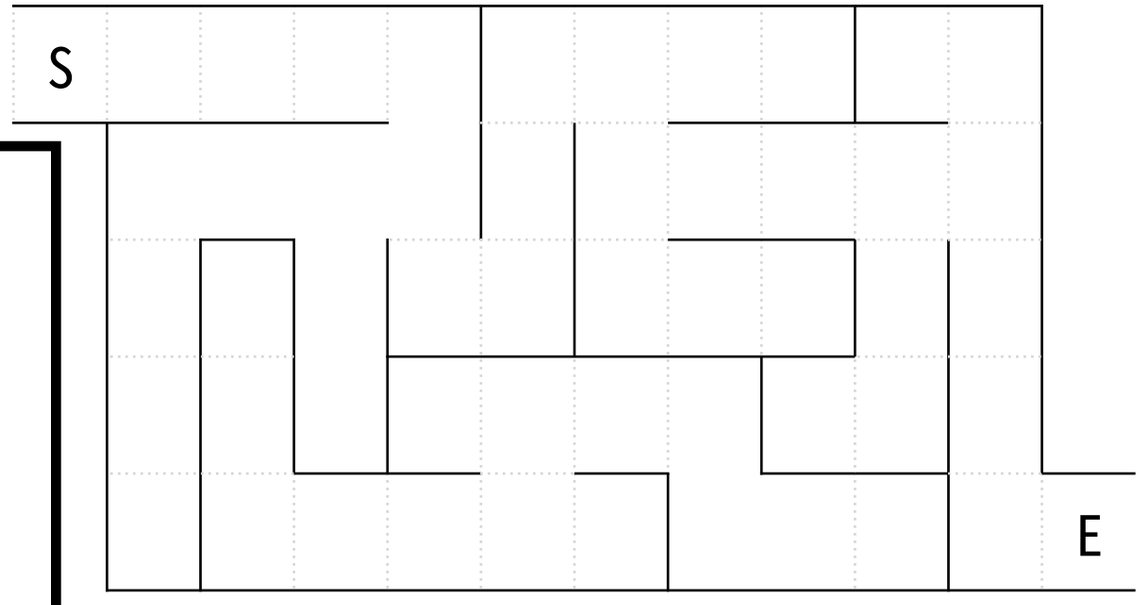


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

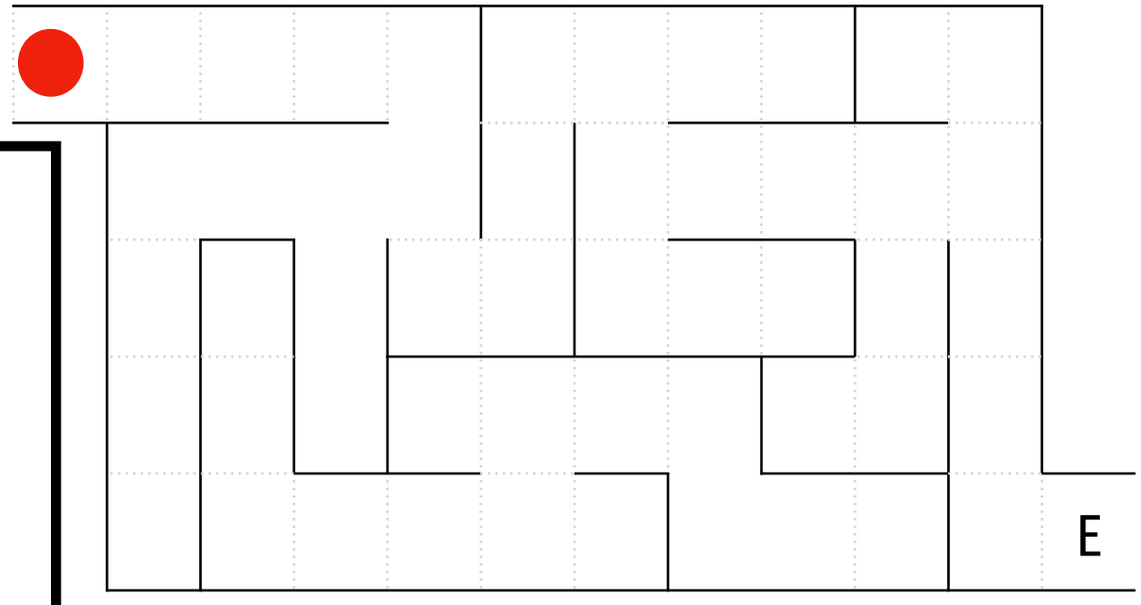


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

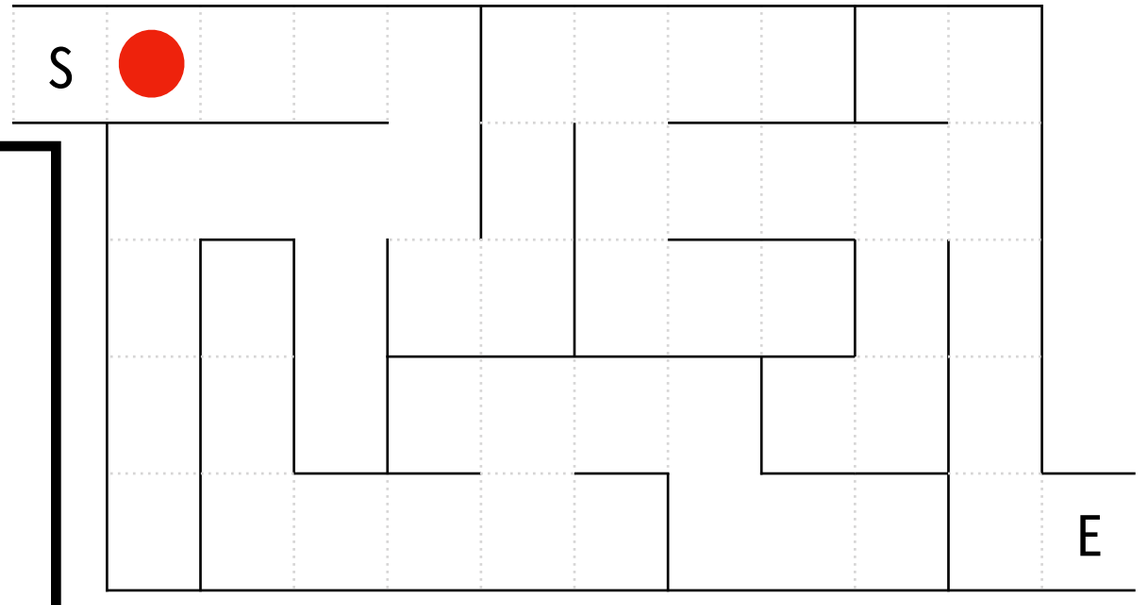


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

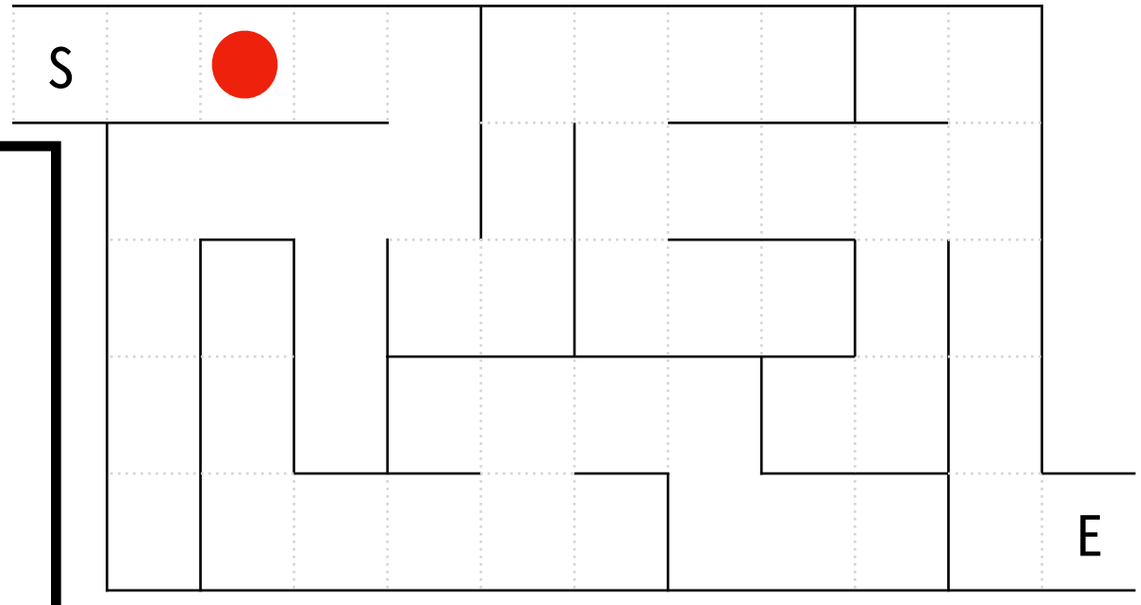


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

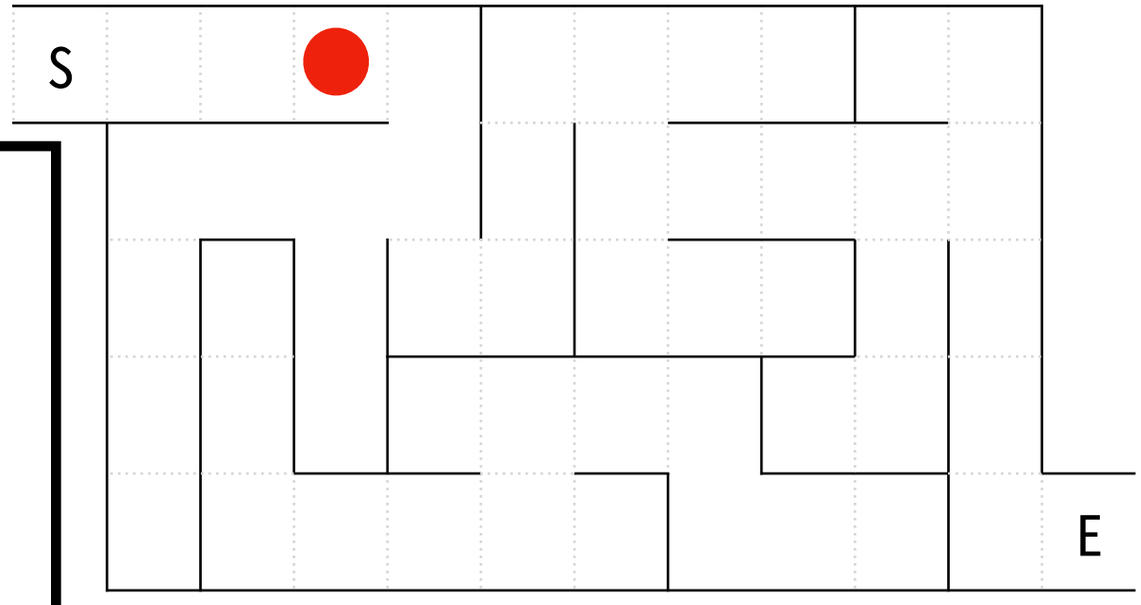


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

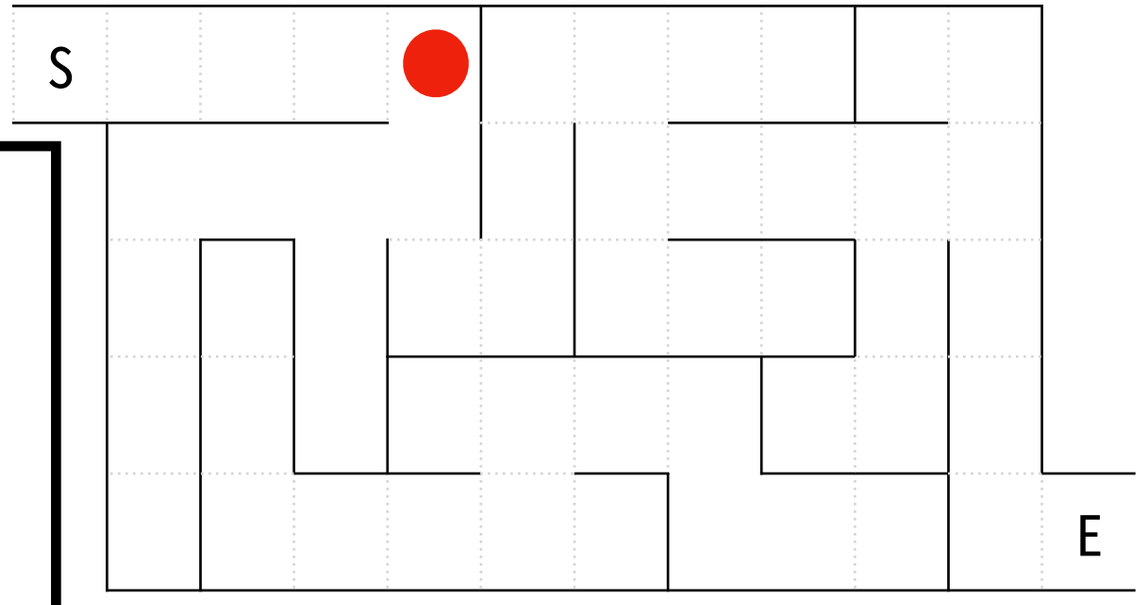


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

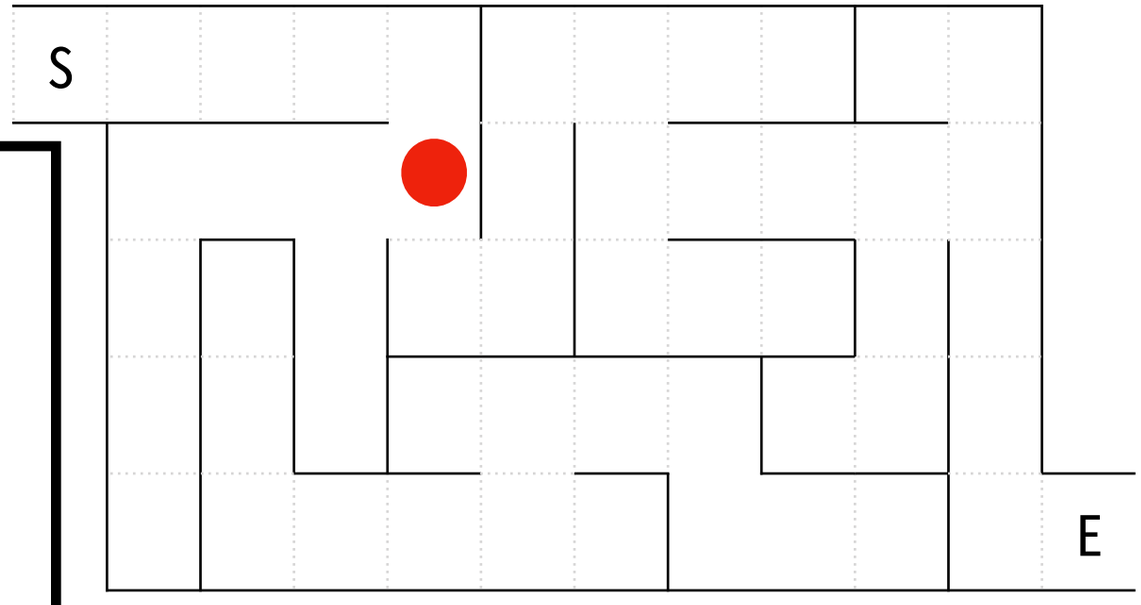


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

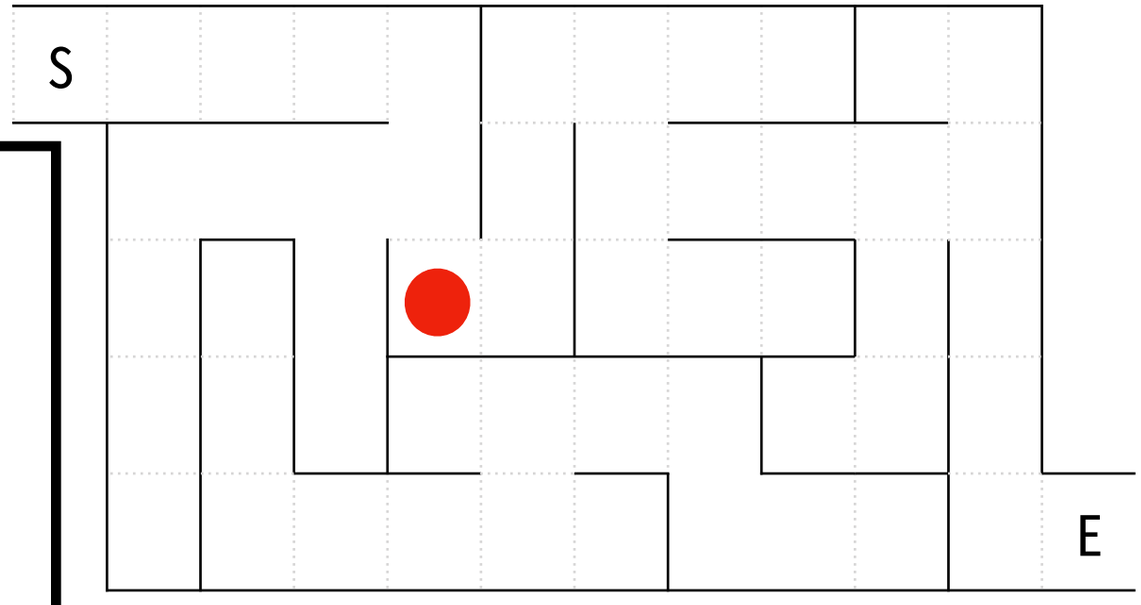


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

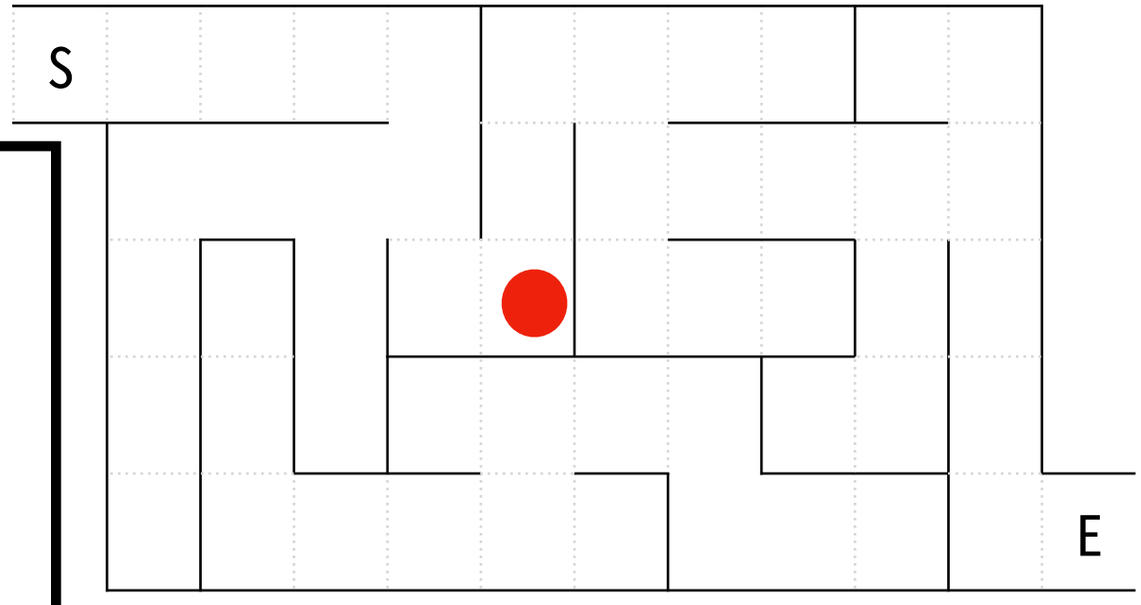


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

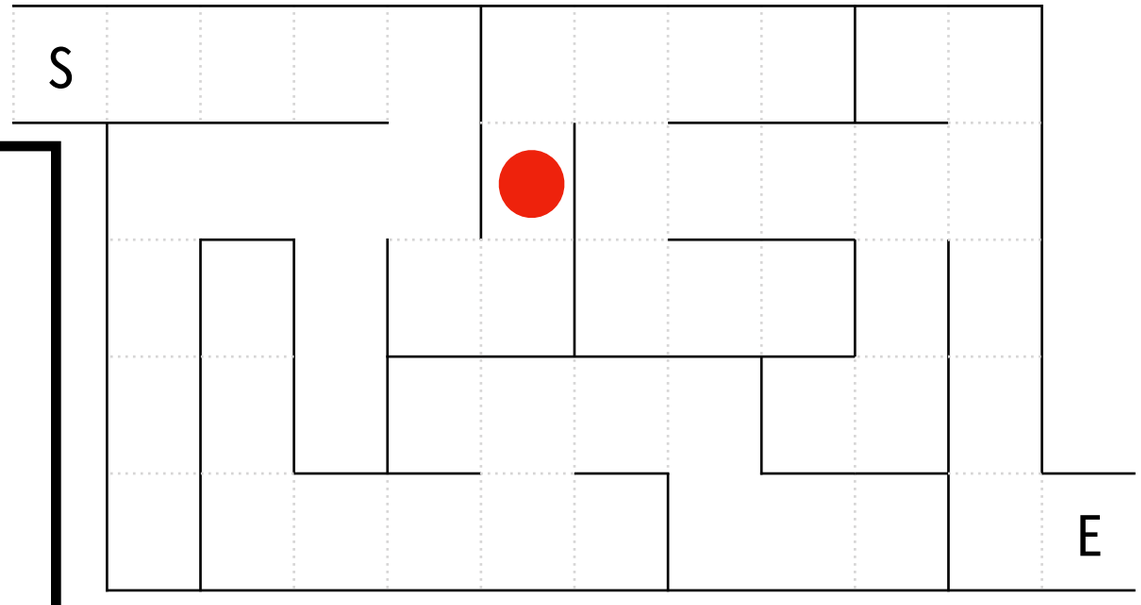


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

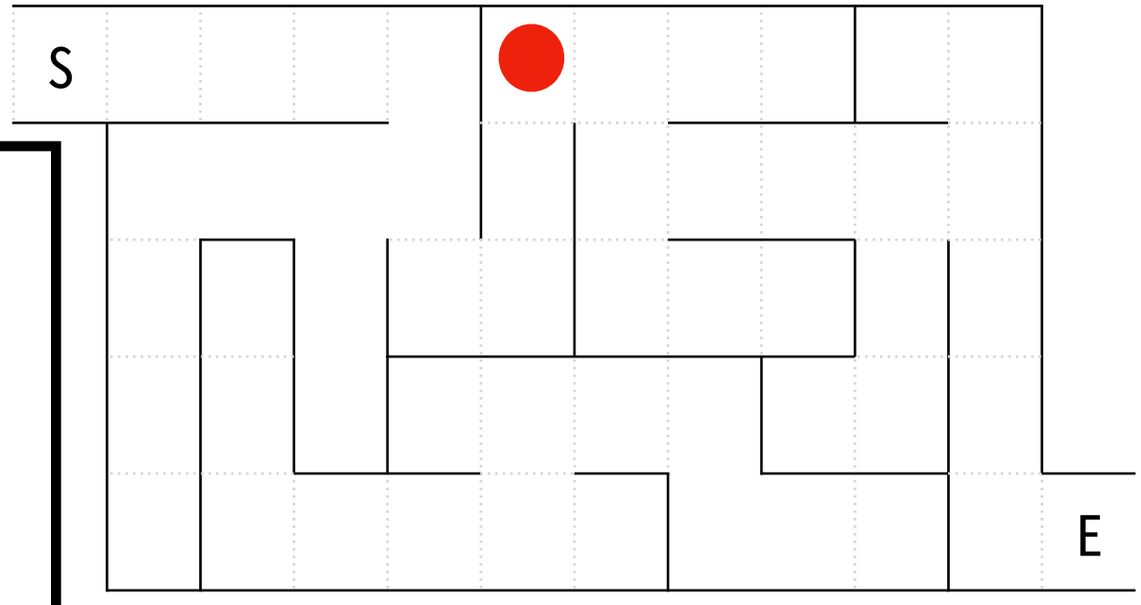


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

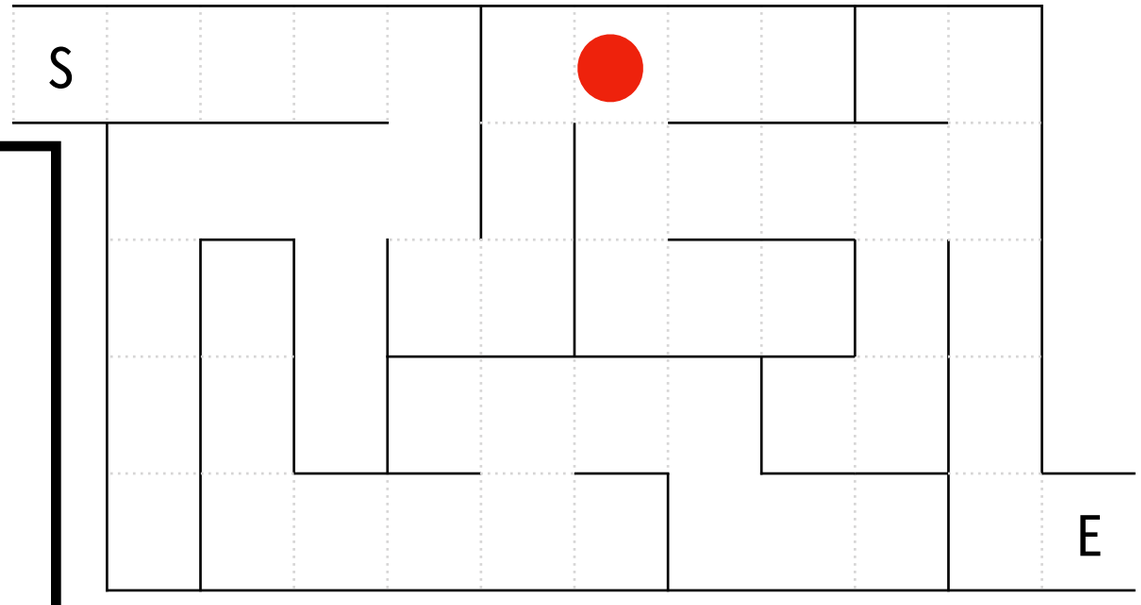


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

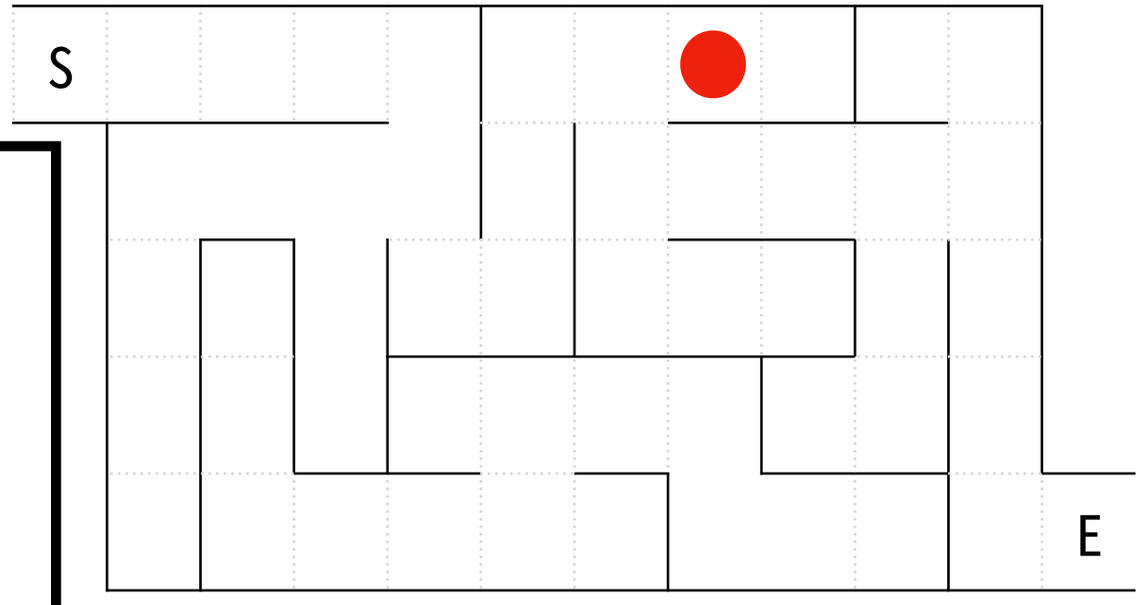


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

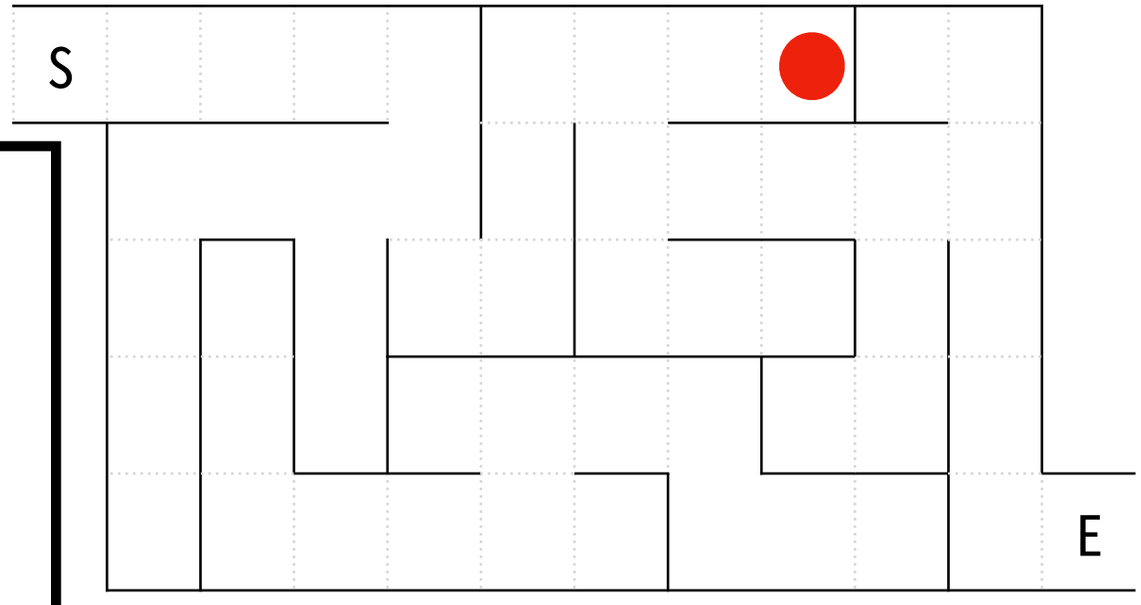


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

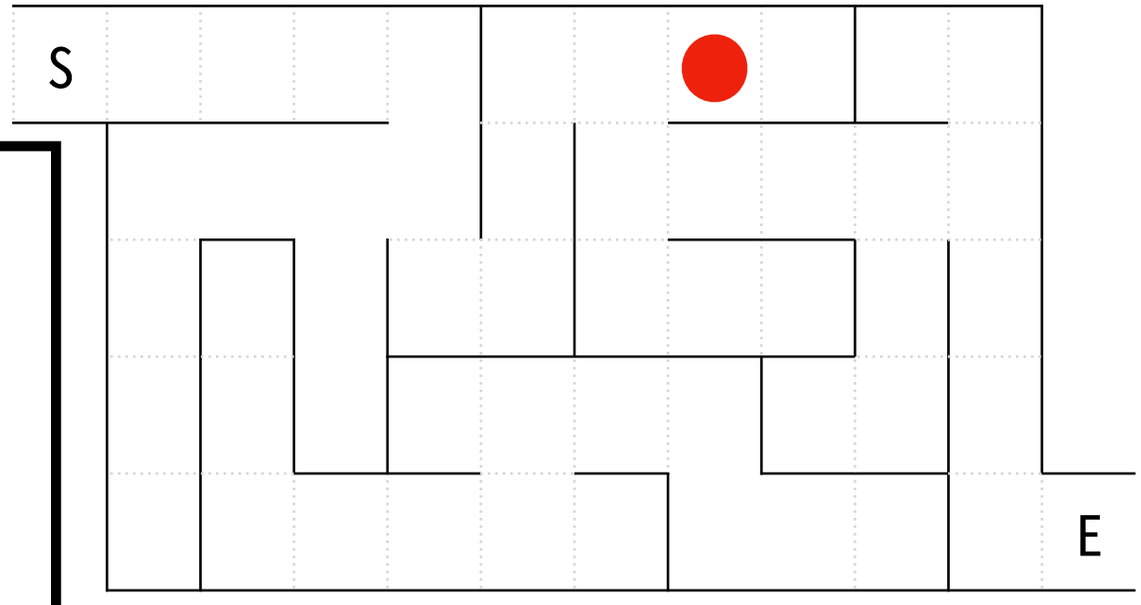


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

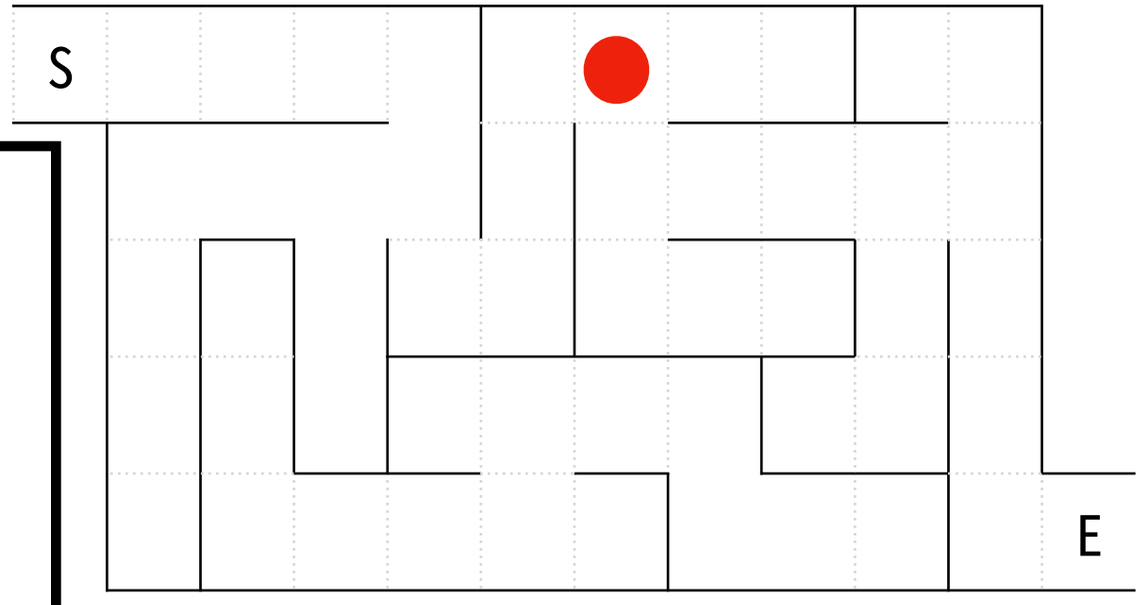


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

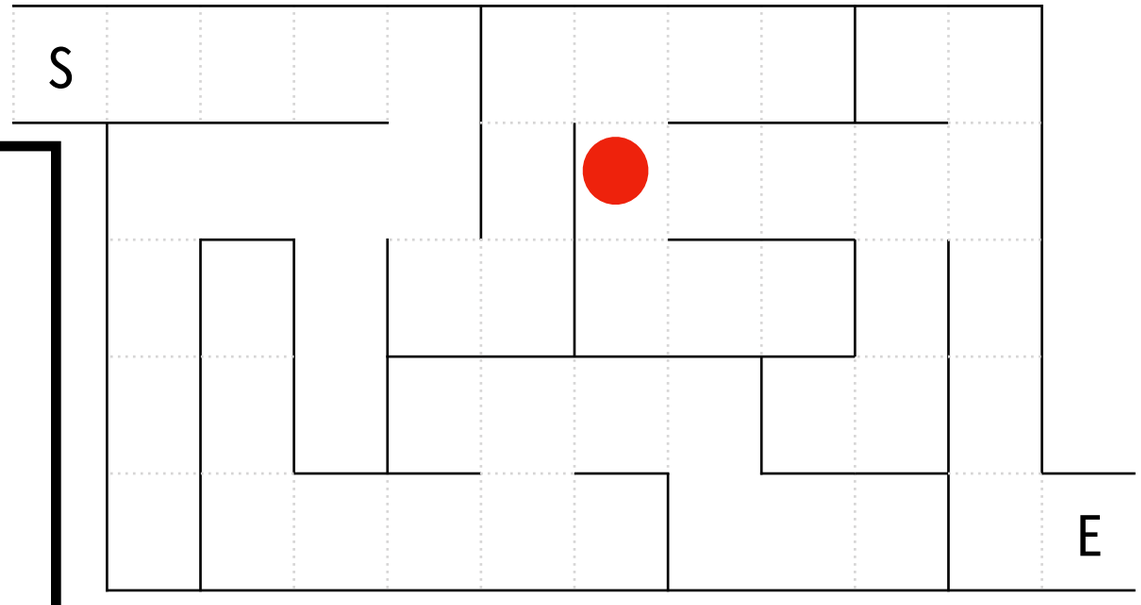


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

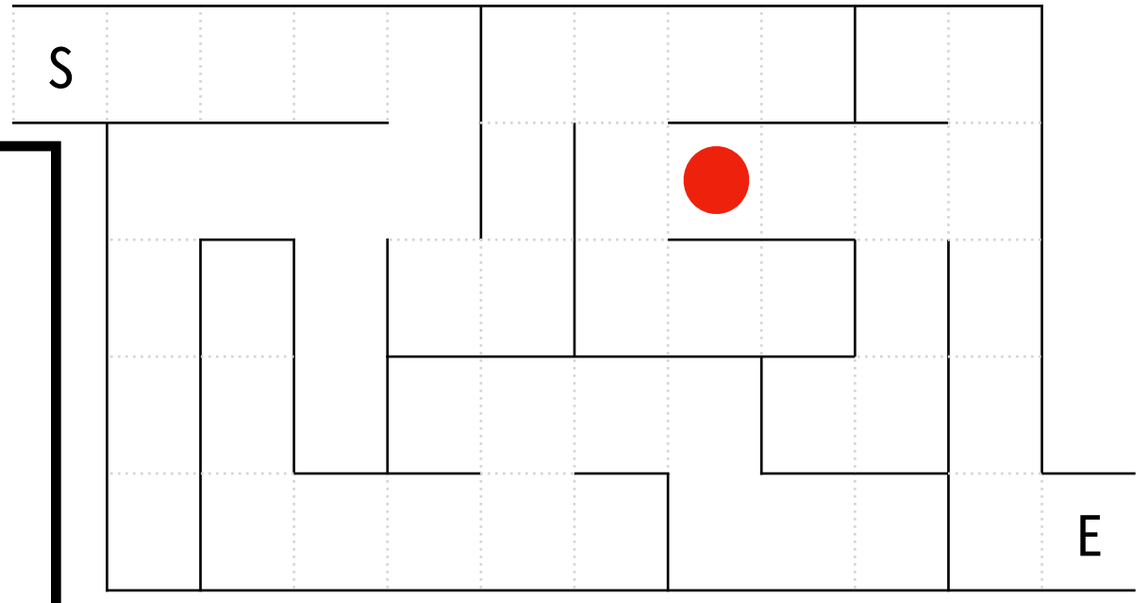


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

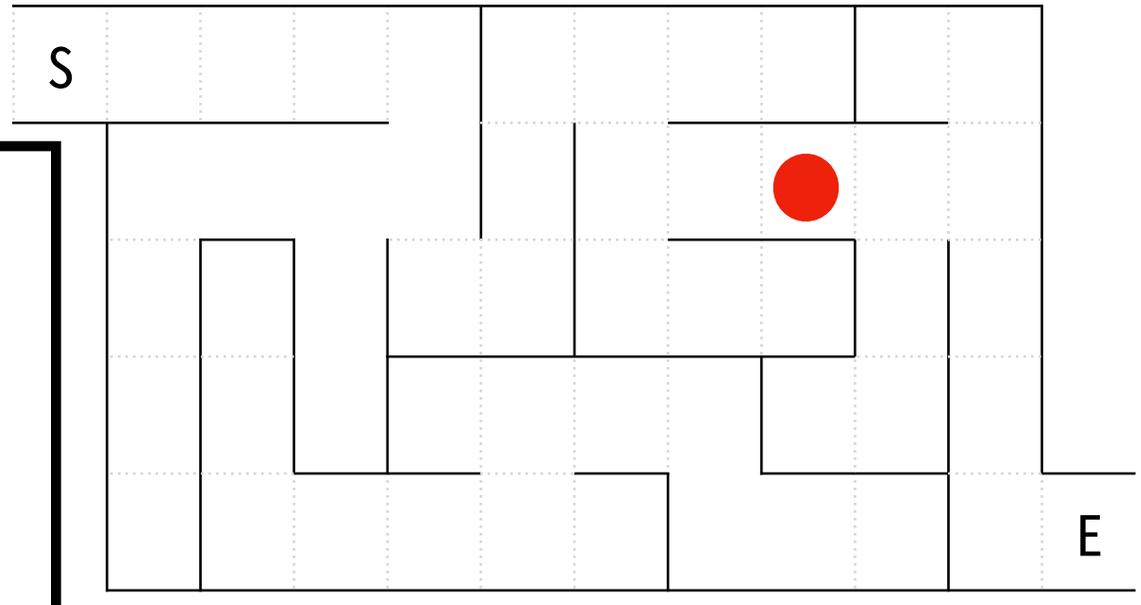


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

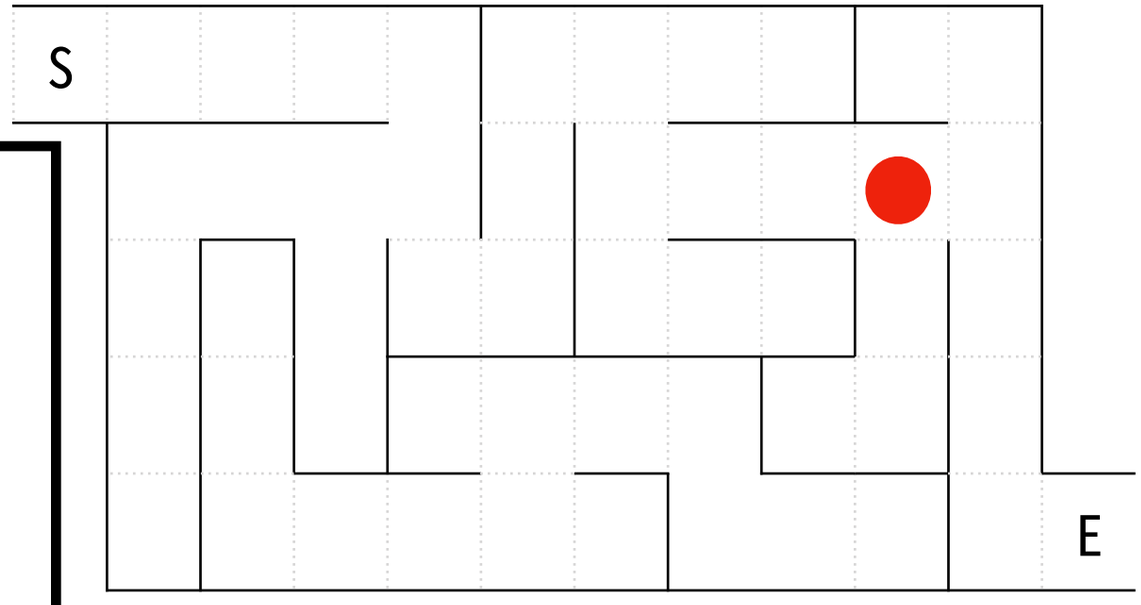


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

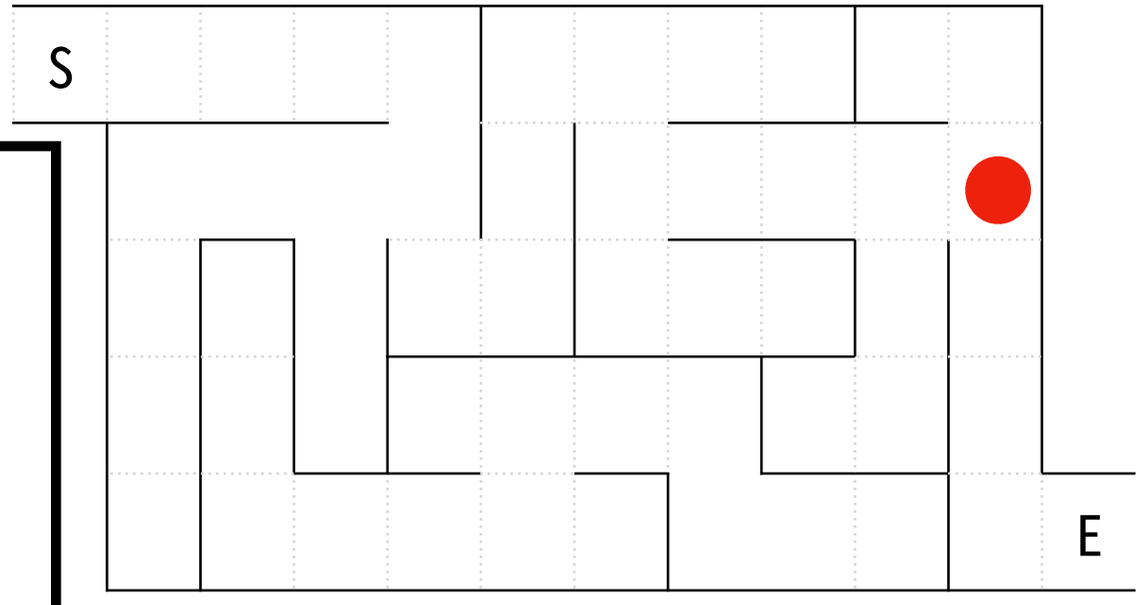


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

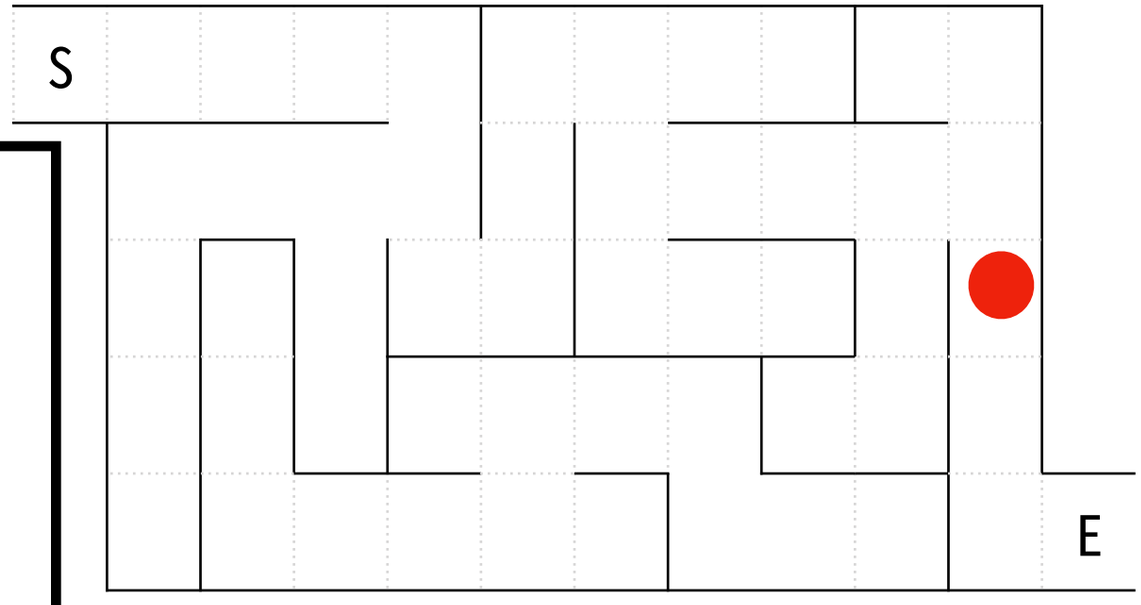


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

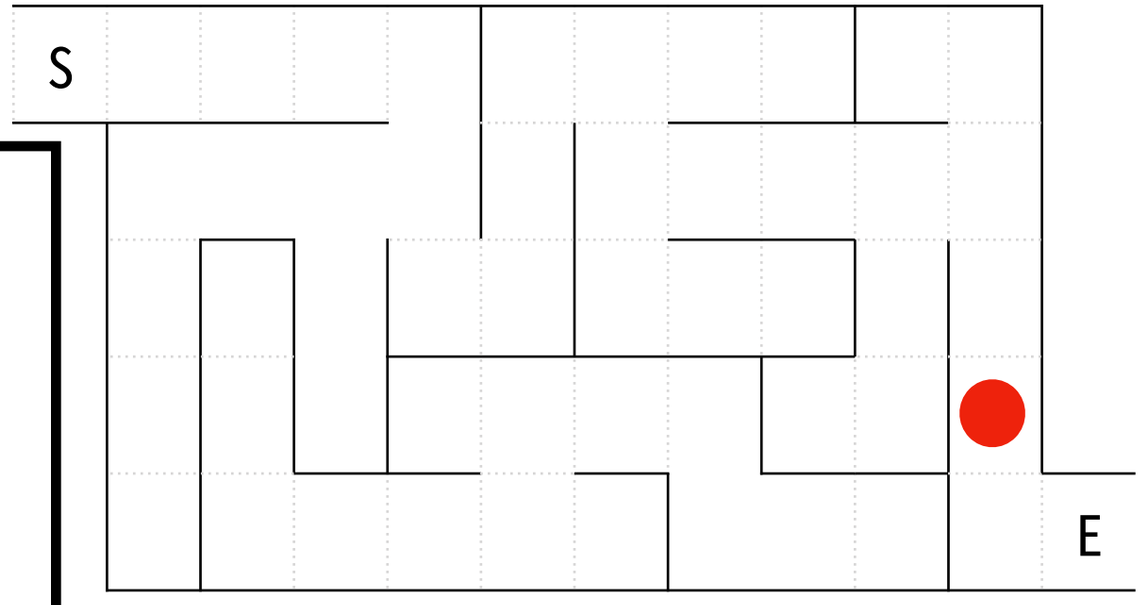


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

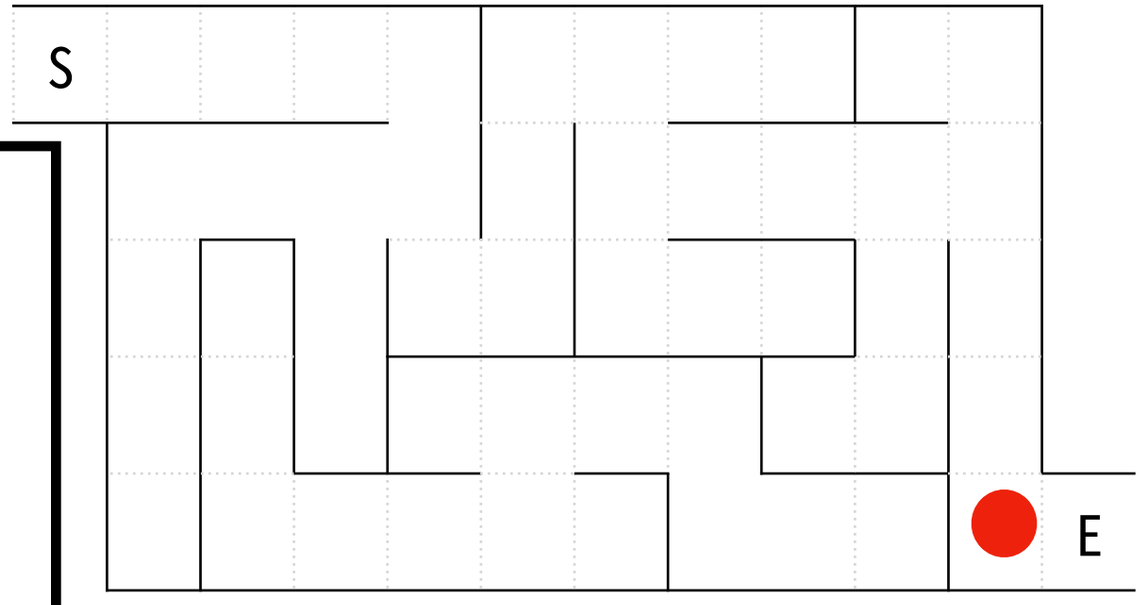


Maze

Depth-First Search

深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```

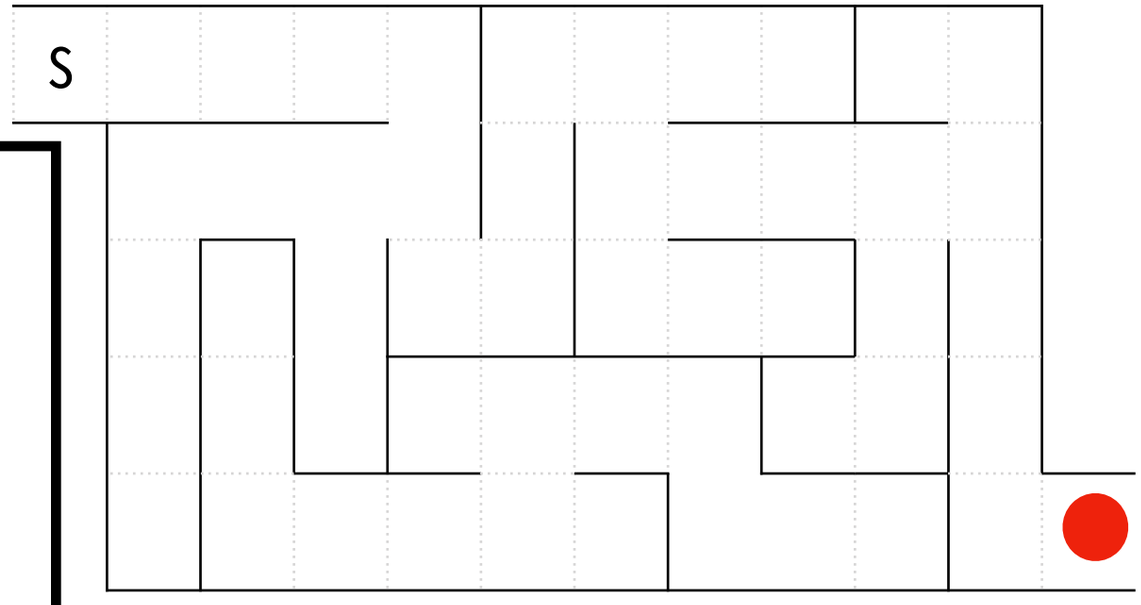


Maze

Depth-First Search

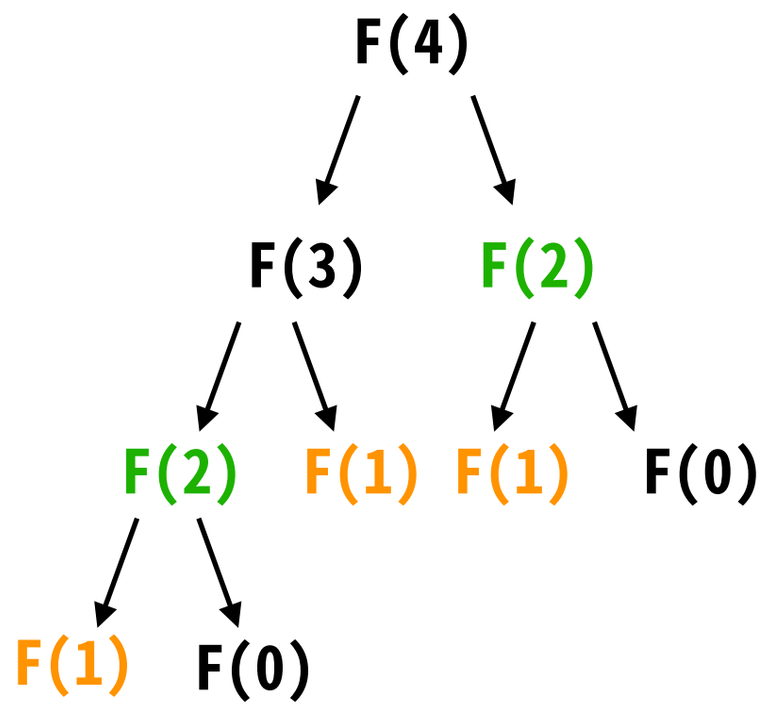
深度優先搜尋

```
void search(int x, int y)
{
    if(x == endX && y == endY)
        return;
    search(x,y+1);    //Right
    search(x+1,y);    //Down
    search(x,y-1);    //Left
    search(x-1,y);    //Up
}
//迷宮需要額外檢查牆壁、路是否已經走過(stack)
```



Maze

再看一次...



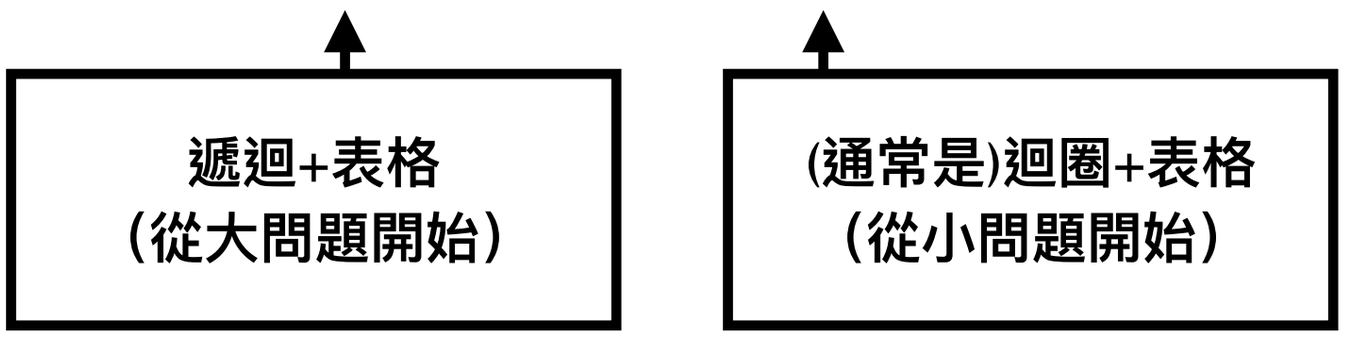
- 重複的事情一直做
- 規模夠大時：浪費時間

動態規劃

- 遞迴產生的子問題一直重複出現：把重複的東西記錄下來，只要再次碰到就查表。

- 表格很重要!!

- 兩種模式：Top-Down & Bottom-Up



組合數與巴斯卡三角形

組合數與巴斯卡三角形

- C(n,k) = n! / (n-k)!k!

組合數與巴斯卡三角形

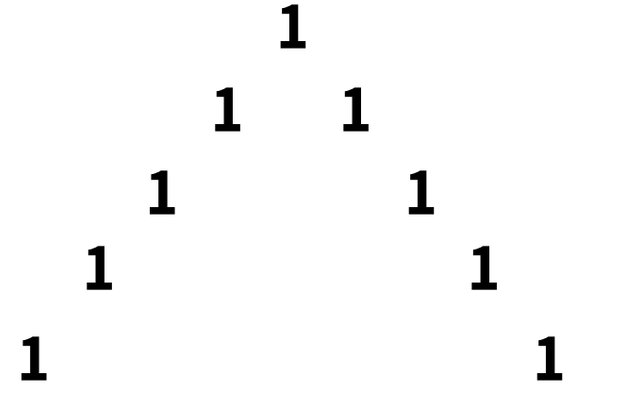
- $C(n,k) = n! / (n-k)!k!$
- $C(n,k) = C(n-1,k-1) + C(n-1,k)$

組合數與巴斯卡三角形

- C(n,k) = n! / (n-k)!k! **Overflow!**

- C(n,k) = C(n-1,k-1) + C(n-1,k)

組合數與巴斯卡三角形

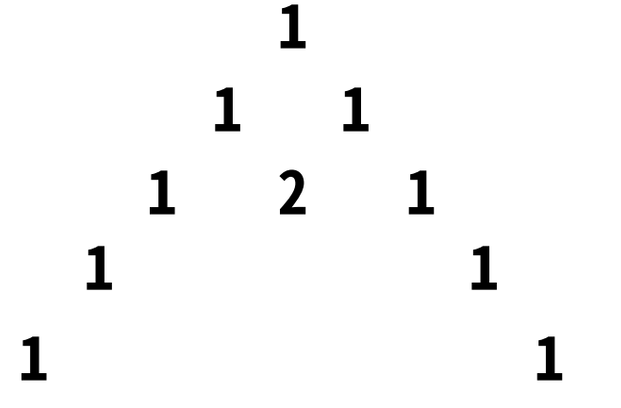


- C(n,k) = n! / (n-k)!k!

Overflow!

- C(n,k) = C(n-1,k-1) + C(n-1,k)

組合數與巴斯卡三角形

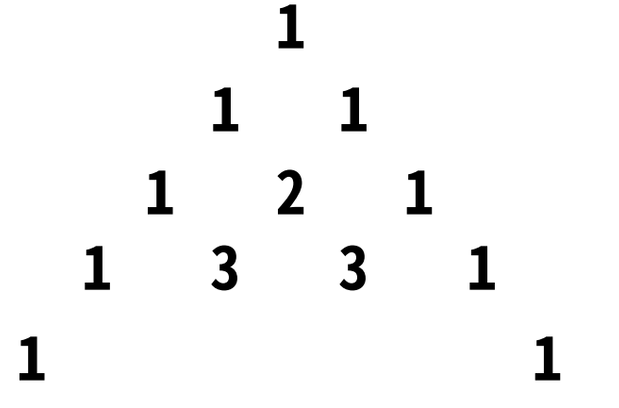


- C(n,k) = n! / (n-k)!k!

Overflow!

- C(n,k) = C(n-1,k-1) + C(n-1,k)

組合數與巴斯卡三角形



- C(n,k) = n! / (n-k)!k!

Overflow!

- C(n,k) = C(n-1,k-1) + C(n-1,k)

組合數與巴斯卡三角形

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1                   1
```

- C(n,k) = n! / (n-k)!k!

Overflow!

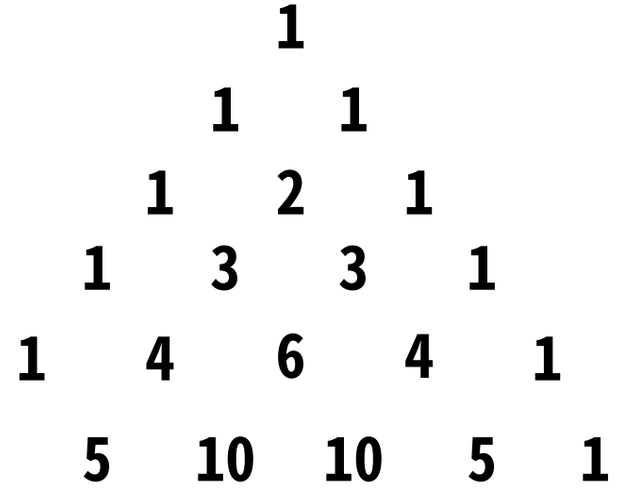
- C(n,k) = C(n-1,k-1) + C(n-1,k)

組合數與巴斯卡三角形

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
```

- C(n,k) = n! / (n-k)!k! **Overflow!**
- C(n,k) = C(n-1,k-1) + C(n-1,k)

組合數與巴斯卡三角形



- C(n,k) = n! / (n-k)!k! **Overflow!**
- C(n,k) = C(n-1,k-1) + C(n-1,k)

n\k	0	1	2	3	4
0	**1**	-	-	-	-
1	**1**	**1**	-	-	-
2	**1**	**2**	**1**	-	-
3	**1**	**3**	**3**	**1**	-
4	**1**	**4**	**6**	**4**	**1**

DP的其它應用

- DAG(Directed Acyclic Graph)
- Rod-Cutting（木棒切割）
- 0/1 背包問題
- 數學問題（Factorial、Fibonacci...）