

w13-Lab

Recursion

Part III

for 204111

by Kittipitch Kuptavanich

General Structure

```


output recurse(arguments) {
    // base case = terminate
    // ถ้าปัญหาเล็กพอที่จะ solve ได้ - ไม่จำเป็นต้องแบ่งอีกต่อไป)
    if smallEnough(arguments)
        return answer

    // divide and conquer (แบ่งปัญหาและเรียกใช้ function ตัวเอง)
    myWorkLoad = someFunction(arguments)
    answerFromSubproblem = recurse(smallerArguments)

    // combine (นำคำตอบมารวมกัน)
    answer = combine(myWorkLoad,answerFromSubproblem);

    return answer
}

```


 ต้องแบ่งแล้วปัญหาเล็กลง หรือซับซ้อน
 น้อยลง และวิ่งเข้าสู่ base case

Adapted From:

<http://ocw.mit.edu/courses/civil-and-environmental-engineering/1-00-introduction-to-computers-and-engineering-problem-solving-spring-2012>

Fibonacci Revisited

```
02 def fib(n):
03     if n == 0 or n == 1:
04         # Base case: fib(0) and fib(1) are both 1
05         return 1
06     else:
07         # Recursive case: fib(n) = fib(n-1) + fib(n-2)
08         return fib(n - 1) + fib(n - 2)
09
10 for n in range(15):
11     print(fib(n), end=" ")
```

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610

Fibonacci Revisited [2]

```

09 def fib(n, depth=0):
10     print("    " * depth,
11           "fib(", n, ")", sep="")
12     if (n < 2):
13         result = 1
14     else:
15         result = fib(n - 1, depth + 1) + \
16                 fib(n - 2, depth + 1)
17     print("    " * depth,
18           "--> ", result, sep="")
19     return result
20
21 fib(4)

```

```

fib(4)
  fib(3)
    fib(2)
      fib(1)
        --> 1
      fib(0)
        --> 1
    --> 2
  fib(1)
    --> 1
--> 3
  fib(2)
    fib(1)
      --> 1
    fib(0)
      --> 1
  --> 2
--> 5

```

ปัญหาบางลักษณะไม่เหมาะ
กับวิธีแก้ปัญหแบบ recursion
(สังเกตการคำนวณซ้ำ)

Fibonacci Memoized

- สามารถหลีกเลี่ยงการคำนวณซ้ำ
- โดยการบันทึกค่า `fib(n)` ที่เคยคำนวณแล้วลงตาราง (Look up table)
- ฟังก์ชันจะเช็คค่าที่บันทึกไว้ในตารางก่อน
- คำนวณใหม่เฉพาะกรณีไม่พบในตาราง
- Top-Down Approach

```

06 def fib(n, table=None):
07     if not table:
08         table = {}
09     if n in table:
10         print("HIT", n)
11         return table[n]
12
13     if n == 0 or n == 1:
14         result = 1
15     else:
16         result = (fib(n - 1, table) +
17                 fib(n - 2, table))
18
19     table[n] = result
20     return result

```



Fibonacci Iterated

- เราสามารถเปลี่ยนวิธีคำนวณแบบ Memoized แบบ top-down
- เป็นการคำนวณแบบล่างขึ้นบน (Bottom-up)
- ไม่ใช้ recursion
- Dynamic Programming
(*Programming* = Planning or Optimizing)

More in Algorithm Class

```

09 def fib(n):
10     table = [0] * (n + 1)
11     table[0] = 1
12     table[1] = 1
13     for i in range(2, n + 1):
14         table[i] = (table[i - 1] +
15                     table[i - 2])
16
17     return table[n]

```

No table

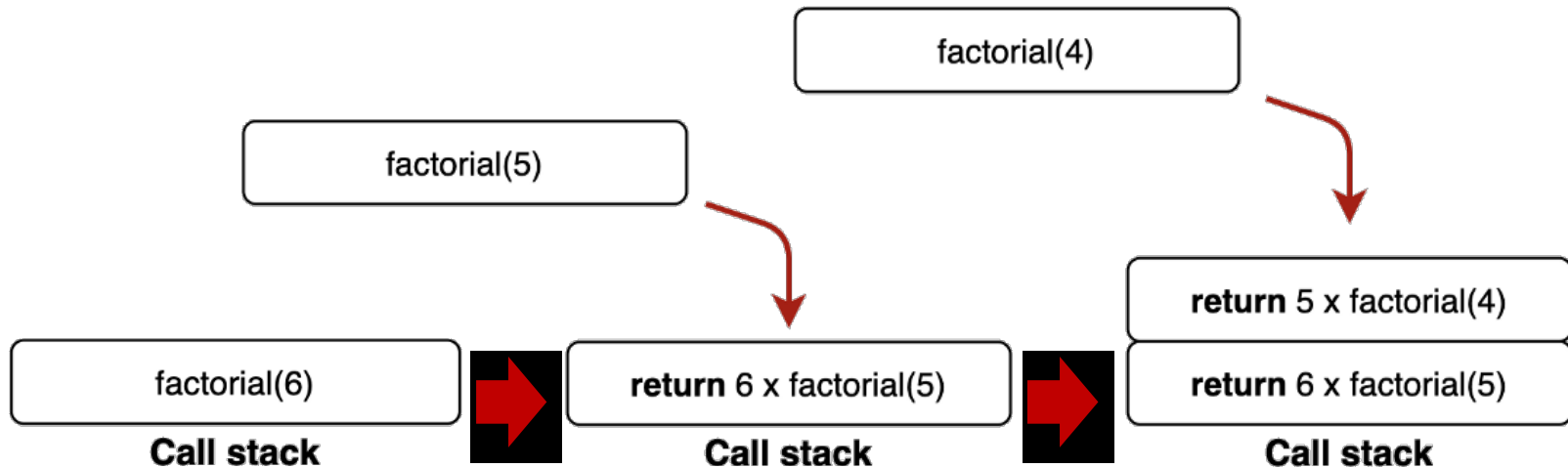
```

08 def fib(n):
09     n_min1 = 1
10     n_min2 = 1
11     result = 1
12     for i in range(2, n + 1):
13         result = n_min1 + n_min2
14         n_min2 = n_min1
15         n_min1 = result
16     return result

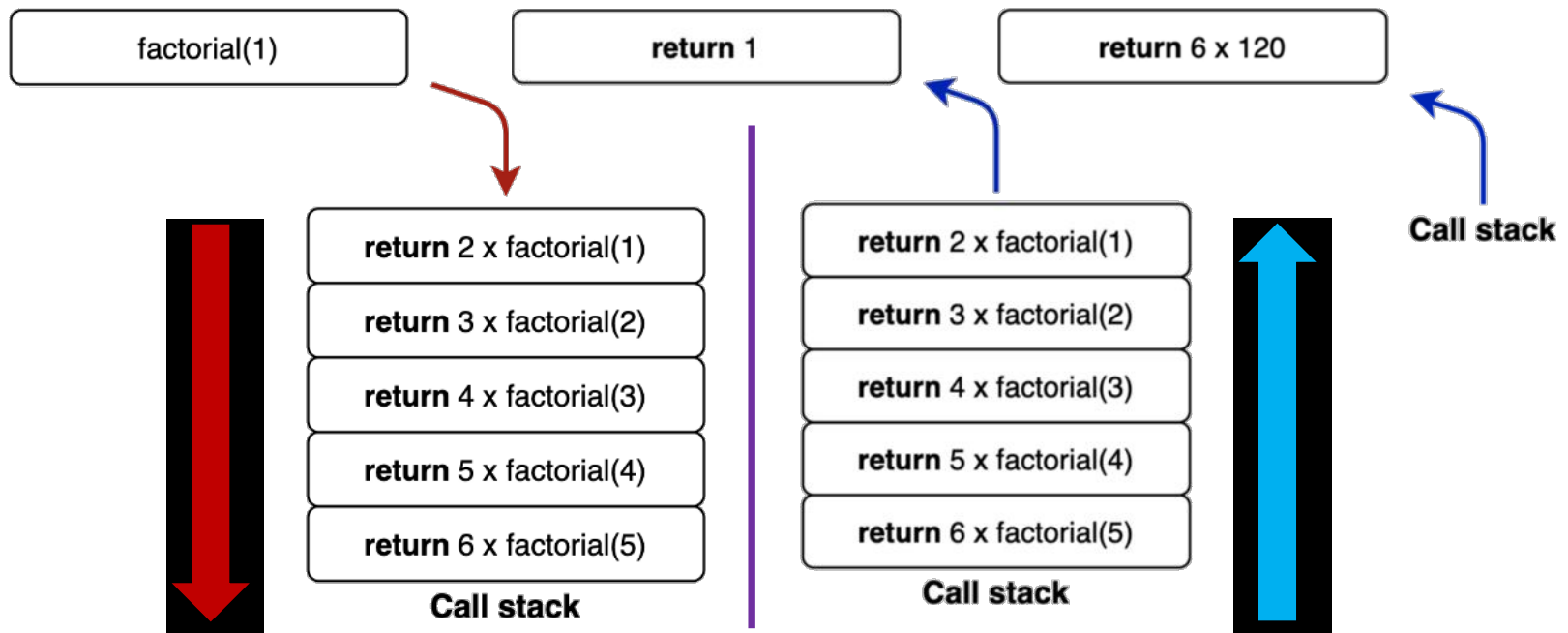
```

Recursion Call Stack

```
function factorial(num) {  
  if (num === 1) return 1;  
  
  return num * factorial(num - 1);  
}
```



Recursion Call Stack [2]



- Stack size can grow and eat up memory if n is large and the problem size is reduced linearly

Iteration vs Recursion Example

Iterative

```
def factorial(n):  
    factorial = 1  
    for i in range(2, n + 1):  
        factorial *= i  
    return factorial  
  
print(factorial(5))
```

Recursive

```
def factorial(n):  
    if (n < 2):  
        return 1  
    else:  
        return n * factorial(n-1)  
  
print (factorial(5))
```

Iteration vs Recursion Example [2]

Iterative

```
def reverse(s):  
    reverse = ""  
    for ch in s:  
        reverse = ch + reverse  
    return reverse  
  
print(reverse("abcd"))
```

Recursive

```
def reverse(s):  
    if (s == ""):  
        return ""  
    else:  
        return (reverse(s[1:]) +  
                s[0])  
  
print(reverse("abcd"))
```

Iteration vs Recursion Example [3]

Iterative

```
def gcd(x, y):  
    while (y > 0):  
        r = x % y  
        x = y  
        y = r  
    return x  
  
print(gcd(1024, 360))
```

Recursive

```
def gcd(x, y):  
    if (y == 0):  
        return x  
    else:  
        return gcd(y, x % y)  
  
print(gcd(1024, 360))
```

Iteration vs Recursion Summary

	Recursion	Iteration
Elegance	++	--
Performance	--	++
Debugability	--	++

- **Conclusion (for now):**

Use iteration when practicable. Use recursion when required (for "naturally recursive problems").

When to Use Recursion

- หลีกเลี่ยง การใช้ recursive function เมื่อมีการใช้ local arrays ขนาดใหญ่ (Recursive ใช้ Memory เยอะ)
- เลือกใช้ recursion เมื่อทำให้ลดเวลาการเขียนโปรแกรม หรือ ทำให้โปรแกรมลดความซับซ้อนลงมาก ๆ
- Recursion เหมาะสมกับงานประเภทที่ใช้ Divide-and-conquer Algorithm ในการแก้ปัญหา เช่น Merge sort และ Binary Search

Reference

- <http://www.kosbie.net/cmu/fall-12/15-112/handouts/notes-recursion/notes-recursion.html>
- <https://medium.com/swlh/visualizing-recursion-6a81d50d6c41>