Algorithm Design and Analysis

Week12: Integer Multiplication, Dynamic Programming,

Adisak Supeesun

3 March 2022

Integer Multiplication

Dynamic Programming (Intro.)

Weighted Interval Scheduling

Integer Multiplication

ให้ A และ B เป็นจำนวนเต็ม 2 จำนวนที่มีความยาว n บิต

ต้องการหาผลคูณของ A และ B

สำหรับปัญหานี้เราจะถือว่า
การ $+, -, \times$ กันของตัวเลข 1 บิต
สองตัวใช้เวลา $O(1)$

Ex. $A = 1100,\ B = 1011$

วิธีคูณแบบปกติ

① นำแต่ละบิตของ B ไปคูณกับ A

1 bit ของ B คูณกับ A ใช้เวลา $O(n)$

B มีทั้งหมด n บิต $\therefore$ ใช้เวลารวม $n \times O(n) = O(n^2)$

② นำผลลัพธ์ที่ได้มาบวกกัน

n บิต × n บิต ได้ 2n บิต

$\therefore$ ใช้เวลาบวกรวม $2n \times O(n) = O(n^2)$

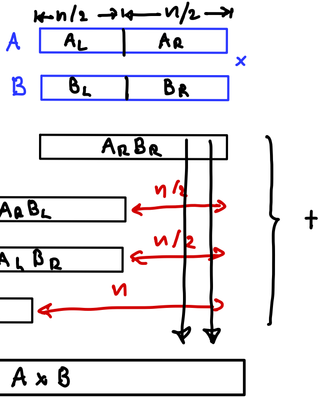
n บิต

$$
\begin{array}{r}
1\,1\,0\,0 \\
\times\quad 1\,0\,1\,1 \\
\hline
1\,1\,0\,0 \\
1\,1\,0\,0 \\
0\,0\,0\,0 \\
1\,1\,0\,0 \\
\hline
1\,0\,0\,0\,0\,1\,0\,0 \\
\hline\hline
\end{array}
$$

n แถว

แต่ละบิตของผลลัพธ์เกิดจากการบวกกัน
ของตัวเลข n บิต $\therefore$ ใช้เวลา $O(n)$

Designing Algorithm



Note $A \times B = (A_L \cdot 2^{n/2} + A_R)(B_L \cdot 2^{n/2} + B_R) = A_L B_L \cdot 2^n + A_L B_R \cdot 2^{n/2} + A_R B_L \cdot 2^{n/2} + A_R B_R$

MUL(A, B)

if $n = 1$

return $A \times B$

end if

$\Big\}$ $O(1)$

divide $\Big\{$

$A_L \leftarrow \lfloor \frac{n}{2} \rfloor$ บิตซ้ายสุดของ A และ $A_R \leftarrow \lceil \frac{n}{2} \rceil$ บิตขวาสุดของ A — $O(n)$

$B_L \leftarrow \lfloor \frac{n}{2} \rfloor$ บิตซ้ายสุดของ B และ $B_R \leftarrow \lceil \frac{n}{2} \rceil$ บิตซ้ายสุดของ B — $O(n)$

conquer $\Big\{$

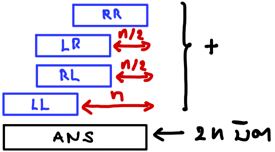
$LL \leftarrow$ MUL(A_L, B_L) — $T(n/2)$

$LR \leftarrow$ MUL(A_L, B_R) — $T(n/2)$

$RL \leftarrow$ MUL(A_R, B_L) — $T(n/2)$

$RR \leftarrow$ MUL(A_R, B_R) — $T(n/2)$

combine $\Big\{$



return ANS

∴ เวลาในการบวก $2n \times O(1) = O(n)$

Running Time

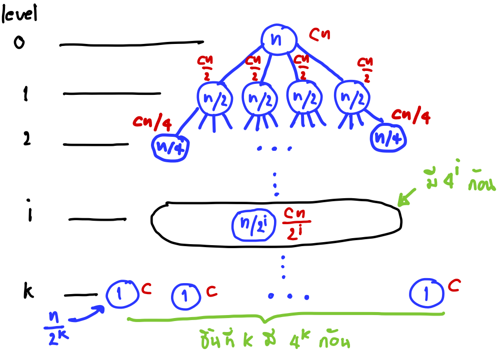
ให้ $T(n)$ เป็นเวลาที่อัลกอริทึมใช้ในการคูณจำนวนเต็มสองจำนวนที่มีความยาว n บิต

จ.น.ปัญหาย่อย

$$T(n) = \begin{cases} 4T\left(\frac{n}{2}\right) + cn & \text{if } n > 1 \\ c & \text{if } n = 1 \end{cases}$$

ขนาดปัญหาย่อย

เวลา divide & combine



cn

$4 \cdot \frac{cn}{2} = 2cn$

$16 \cdot \frac{cn}{4} = 4cn$

$\vdots$

$4^i \cdot \frac{cn}{2^i} = 2^i \cdot cn$

$\vdots$

$4^k \cdot c$

$$T(n) = \Big[\underbrace{2^0 cn + 2^1 cn + 2^2 cn + \dots + 2^{k-1} cn}_{\text{ผลบวกชั้นที่ 0 ถึง } k-1} \Big] + \underbrace{4^k \cdot c}_{\text{ชั้นที่ } k}$$

$$= cn\Big[\underbrace{2^0 + 2^1 + 2^2 + \dots + 2^{k-1}}_{\text{geometric series}} \Big] + 4^k \cdot c$$

$$\therefore \text{มีค่าเท่ากับ } \frac{2^{k-1+1} - 1}{2 - 1} = 2^k - 1$$

$$= cn(2^k - 1) + 4^k \cdot c$$

$$= cn(\underbrace{2^{\log_2 n}}_{n} - 1) + \underbrace{4^{\log_2 n}}_{2^{2\log_2 n}} \cdot c \qquad (\text{แทนค่า } k = \log_2 n)$$

จาก $b^{\log_b n} = n$

หาค่า k

จาก $\frac{n}{2^k} = 1$

$\Rightarrow n = 2^k$

$\Rightarrow \log_2 n = \log_2 2^k$

$= k \cdot \log_2 2 \quad (\log_b a^c = c \log_b a)$

$= k \quad (\log_b b = 1)$

$\therefore k = \log_2 n$

$$= cn(n-1) + \underbrace{2^{2\log_2 n}}_{2^{\log_2 n^2}} \cdot c$$

$(\text{as } c\log_b n = \log_b n^c)$

$$= cn(n-1) + \underbrace{2^{\log_2 n^2}}_{n^2} \cdot c$$

$$= cn(n-1) + cn^2$$

$$= 2cn^2 - cn$$

$$= O(n^2)$$

Logarithm and Geometric Series

- **Logarithm** : เราจะกล่าวว่า $x = \log_b n$ ก็ต่อเมื่อ $b^x = n$

  คุณสมบัติของ logarithm

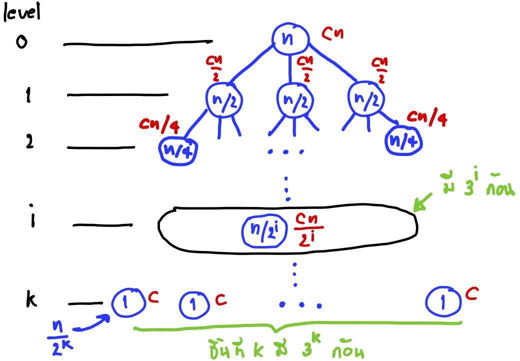
  1. $\log_b b = 1$
  2. $\log_b nm = \log_b n + \log_b m$
  3. $\log_b \frac{n}{m} = \log_b n - \log_b m$
  4. $\log_b n^c = c \log_b n$
  5. $\log_b n = \frac{log_k n}{log_k b}$
  6. $b^{\log_b n} = n$

- **Geometric Series** : สำหรับจำนวนจริง $r \neq 1$ และจำนวนเต็ม $n \geq 0$ ใดๆ

  $$r^0 + r^1 + r^2 + \ldots + r^n = \frac{r^{n+1} - 1}{r - 1}$$

Ex. จงหารูปแบบปิดของ recurrence ต่อไปนี้

$$T(n) = \begin{cases} 3T(\frac{n}{2}) + cn & \text{if} \quad n > 1 \\ c & \text{if} \quad n = 1 \end{cases}$$



cn

$3 \cdot \frac{cn}{2} = cn$

$9 \cdot \frac{cn}{4} = cn$

$\vdots$

$3^i \cdot \frac{cn}{2^i} = \cdot cn$

$\vdots$

$3^k \cdot c$

$1 = \frac{n}{2^k} \quad \therefore k = \log_2 n$

$$T(n) = \Big[\underbrace{\left(\tfrac{3}{2}\right)^0 cn + \left(\tfrac{3}{2}\right)^1 cn + \ldots + \left(\tfrac{3}{2}\right)^{k-1} cn}_{\text{ผลบวกชั้นที่ 0 ถึง } k-1} \Big] + \underbrace{c \cdot 3^k}_{\text{ชั้นที่ } k}$$

$$= cn \Big[\underbrace{\left(\tfrac{3}{2}\right)^0 + \left(\tfrac{3}{2}\right)^1 + \ldots + \left(\tfrac{3}{2}\right)^{k-1}}_{\text{geometric series}} \Big] + c \cdot 3^k$$

$$\therefore \text{ มีค่าเท่ากับ } \frac{\left(\frac{3}{2}\right)^{k-1+1} - 1}{\left(\frac{3}{2}\right) - 1} = 2\left(\left(\tfrac{3}{2}\right)^k - 1\right)$$

$$= 2cn\Big(\underbrace{\left(\tfrac{3}{2}\right)^k - 1}_{\leq \left(\frac{3}{2}\right)^k}\Big) + c \cdot 3^k$$

$$\leq 2cn\left(\tfrac{3}{2}\right)^k + c \cdot 3^k$$

แทน k ด้วย $\log_2 n$ จะได้

$$T(n) \leq 2cn\underbrace{\left(\frac{3}{2}\right)^{\log_2 n}}_{\frac{3^{\log_2 n}}{\cancel{2^{\log_2 n}} \to n}} + c \cdot 3^{\log_2 n}$$

(จาก $b^{\log_b n} = n$)

$$= 2c\cancel{n} \cdot \frac{3^{\log_2 n}}{\cancel{n}} + c \cdot 3^{\log_2 n}$$

$$= 3c \cdot 3^{\log_2 n} \longrightarrow 3^{\log_2 n} = \left(2^{\log_2 3}\right)^{\log_2 n} = \left(\cancel{2^{\log_2 n}}\right)^{\log_2 3} = n^{\log_2 3}$$

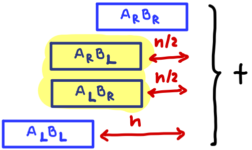
$3 = 2^{\log_2 3}$, $2^{\log_2 n} = n$

$$= 3cn^{\log_2 3}$$

$$< 3cn^{1.59}$$ (เนื่องจาก $\log_2 3 < 1.59$)

$$\therefore T(n) = O(n^{1.59})$$

โดยเราต้อง recursive 4 ครั้งเพื่อหาผลคูณ A_RB_R , A_RB_L , A_LB_R และ A_LB_L
จากนั้นนำผลลัพธ์ที่ได้มาบวกกันดังนี้



จาก $(A_L+A_R)(B_L+B_R)$

$$= A_LB_L + A_LB_R + A_RB_L + A_RB_R$$

เราสามารถหาผลรวมกลางทั้งสองได้โดยการ recursive เพียง 3 ครั้งได้ดังนี้

(1) หา A_RB_R , (2) หา A_LB_L และ (3) หา $(A_L+A_R)(B_L+B_R)$

$\frac{n}{2}+1$ บิต

และสามารถหา $A_RB_L + A_LB_R$ ได้จาก (3) − (1) − (2)

MUL(A, B)

if $n = 1$

  return $A \times B$

end if

$\Big\}$ $O(1)$

divide $\begin{cases} A_L \leftarrow \lfloor \frac{n}{2} \rfloor \text{ บิตซ้ายสุดของ } A \text{ และ } A_R \leftarrow \lceil \frac{n}{2} \rceil \text{ บิตขวาสุดของ } A \\ B_L \leftarrow \lfloor \frac{n}{2} \rfloor \text{ บิตซ้ายสุดของ } B \text{ และ } B_R \leftarrow \lceil \frac{n}{2} \rceil \text{ บิตซ้ายสุดของ } B \end{cases}$ $O(n)$

conquer $\begin{cases} LL \leftarrow \text{MUL}(A_L, B_L) & \text{——} \; T(\frac{n}{2}) \\ RR \leftarrow \text{MUL}(A_R, B_R) & \text{——} \; T(\frac{n}{2}) \\ X \leftarrow \text{MUL}(A_L + A_R, B_L + B_R) & \text{——} \; T(\frac{n}{2}+1) \end{cases}$

combine:

LRRL $\leftarrow$ X - LL - RR

		RR
	LRRL	$n/2$
LL	n	
ANS		

$+$

ผลลัพธ์มี 2n บิต
แต่ละบิตเกิดจากการ
บวกกันของ 3 บิต
∴ ใช้เวลาบวกทั้งหมด

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

return ANS

รวมเวลา

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + T(\frac{n}{2}+1) + cn & \text{if } n > 1 \\ c & \text{if } n = 1 \end{cases}$$

ซึ่งสามารถคิดแบบปัดได้เป็น

$$T(n) = \begin{cases} 3T(\frac{n}{2}) + cn & \text{if } n > 1 \\ c & \text{if } n = 1 \end{cases}$$

Fibonacci Sequence

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

- $F(i) = \begin{cases} F(i-1) + F(i-2) & \text{if} \quad i > 2 \\ 1 & \text{if} \quad i = 1 \text{ or } i = 2 \end{cases}$

- recursive algorithm

```
Fibo(i)
   if n=1 or n=2
      return 1
   end if
   return Fibo(i-1) + Fibo(i-2)
```

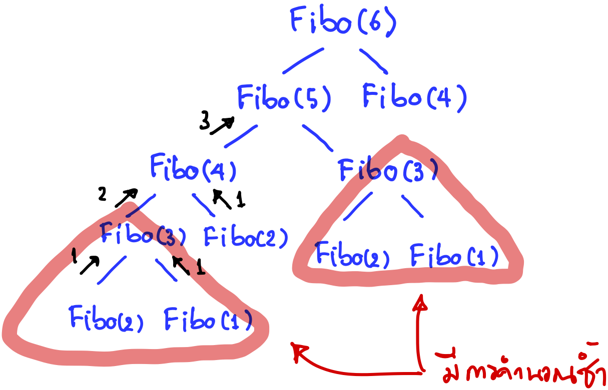
$\left.\begin{array}{l} \text{if } n=1 \text{ or } n=2 \\ \quad \text{return } 1 \end{array}\right\} O(1)$

เวลาในการทำงาน

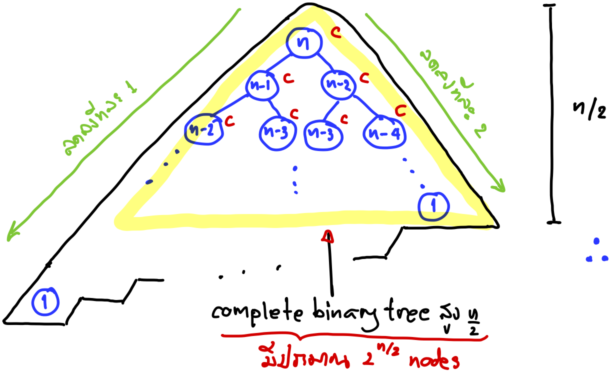
ให้ T(n) แทนเวลาที่ฟังก์ชัน Fibo
ใช้ในการหาค่า F(n)

$\therefore T(n) = \begin{cases} T(n-1) + T(n-2) + c & \text{if } n > 2 \\ c & \text{if } n = 1, 2 \end{cases}$

- เวลาในการทำงานของ recursive algorithm



หาก $T(n) = \begin{cases} T(n-1) + T(n-2) + c & \text{if } n > 2 \\ c & \text{if } n = 1, 2 \end{cases}$



$\therefore$ เวลาในการทำงานของ Fibo(n)

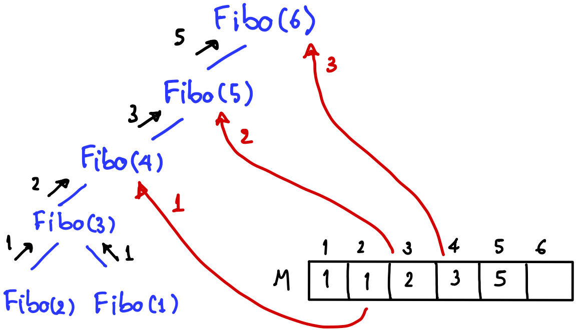
$\geq c \cdot 2^{\frac{n}{2}} = \Omega(2^{\frac{n}{2}})$

(เป็น exponential ของ n)

ไม่มีประสิทธิภาพ !

- Memoization จำคำตอบที่เคยคำนวณไปแล้วไว้ในตาราง

(ก่อน recursive เช็คก่อนว่ามีคำตอบในตารางหรือยัง)

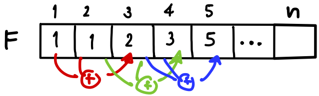


Note การ recursive เป็นการคำนวณแบบ Top-down อยากรู้คำตอบเมื่อไหร่ค่อยไปคำนวณหา

- Bottom-up

สังเกตว่าการเติมตารางคำตอบของ recursive + memoization จะค่อยๆ เติมคำตอบทีละช่อง เริ่มจากช่องที่มี index น้อย

เราสามารถทำการเติมตารางแบบเดียวกันได้โดยการวนลูป (ไม่ต้อง recursive)




```
F[1] ← 1 , F[2] ← 1
for i = 3, 4, ..., n
    F[i] ← F[i-1] + F[i-2]
```

ใช้เวลา $O(n)$

Dynamic Programing

- เทคนิคการออกแบบอัลกอริทึมที่มีการนำตารางเข้ามาช่วยในการหาคำตอบ
- สร้างคำตอบของปัญหาที่ใหญ่กว่า จากคำตอบของปัญหาที่เล็กกว่า ซึ่งถูกบันทึกลงในตารางจากการคำนวณก่อนหน้านั้น
- คิดแบบ recursive แต่คำนวณแบบ bottom up