Department of Computer and Information Science

การวิเคราะห์ประสิทธิภาพของอัลกอริทึม (2)

อ. ลือพล พิพานเมฆาภรณ์

ขั้นตอนการวิเคราะห์สำหรับอัลกอริทึมแบบ non-recursive

1. ระบุขนาดของอินพุต (input size) และ basic operation ของอัลกอริทึม
    - basic operation มักอยู่ในลูปในสุด (most nested loop)

2. จากขนาดของอินพุต พิจารณาว่าอัลกอริทึมจำเป็นต้องวิเคราะห์กรณีเลวร้ายสุด (worst case) กรณีดีที่สุด (best case) และกรณีเฉลี่ย (average case)
    - จำนวนรอบการทำงานขึ้นอยู่กับลักษณะของข้อมูลหรือไม่

3. เขียนฟังก์ชั่นเวลา T(n) เพื่อนับรอบการทำงานพร้อมแก้สมการ

4. ปรับฟังก์ชั่นเวลาของ basic operation ให้เข้าสู่ฟังก์ชั่นเวลาอ้างอิงที่ใกล้เคียงที่สุดโดยใช้สัญกรเชิงเส้นกำกับ เช่น บิ๊กโอ

สูตร Summation เพื่อช่วยนับรอบการทำงานกรณี for loop

- $\sum_{i=1}^{n} 1 = 1 + 1 + \cdots + 1 = n \in \Theta(n)$
- $\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \in \Theta(n^2)$
- $\sum_{i=1}^{n} i^k = 1 + 2^k + \cdots + n^k \approx \frac{n^{k+1}}{k+1} \in \Theta(n^{k+1})$
- $\sum_{i=1}^{n} a^i = 1 + a + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1} \in \Theta(a^n)$
- $\sum_{i=1}^{n} (a_i \pm b_i) = \sum_{i=1}^{n} a_i \pm \sum_{i=1}^{n} b_i$ $\qquad \sum_{i=1}^{n} c\, a_i = c \sum_{i=1}^{n} a_i$

ตัวอย่าง 1

จงวิเคราะห์เวลาการทำงานของอัลกอริทึมต่อไปนี้ ในรูปของบิ๊กโอ (Big-Oh) บิ๊กโอเมก้า (Big-Omega) และ บิ๊กเทต้า (Big-Theta)

```
MaxElement (A[1...n])
 Maxval:= A[1]
 for i:= 2 to n do
    if A[i] > maxval then
          maxval:= A[i]
return maxval
```

T(n) =

ตัวอย่าง 2

จงวิเคราะห์เวลาการทำงานของอัลกอริทึมต่อไปนี้ ในรูป $\Theta(g(n))$

```
MatrixMultiplication (A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])
for i ← 0 to n-1 do
    for j ← 0 to n-1 do
         C[ i, j ] ← 0.0
          for k ← 0 to n-1 do
               C[ i, j ] ← C[ i, j] + A[ i, k] * B[ k, j]
return C
```

ตัวอย่าง 3

จงประมาณเวลาทำงานของอัลกอริทึม Selection Sort ในรูป O บิ๊กโอ (Big-Theta)

```
Algorithm SelectionSort(A[0..n-1])
  for i←0 to n-2 do
      min←i
      for j←i+1 to n-1 do
          if A[j] < A[min]
              min ← j
      swap A[i] and A[min]
```

Example:

|31 25 12 22 11

11 | 25 12 22 31

11 12 | 25 22 31

11 12 22 | 25 31

11 12 22 25 | 31

ตัวอย่าง 4

จงประมาณเวลาทำงานของอัลกอริทึม Insertion Sort ในรูป O บิ๊กโอ (Big-Theta)

```
Algorithm InsertionSort( A[0 ..N-1] )
  for i←1 to n-1 do
    v ← A[i]
    j ← i-1
    while j≥0 and A[j]>v do
        A[j+1] ← A[j]
        j ← j-1
    A[j+1] ← v
```

Example:

31 |	25	12	22	11
25	31 |	12	22	11
12	25	31 |	22	11
12	22	25	31 |	11

ตัวอย่าง

```
algorithm factorial(n):
    if n=0 then:
        return 1;
    else:
        return factorial(n-1)*n;
end algorithm
```

ขั้นตอนการวิเคราะห์เวลาสำหรับอัลกอริทึมแบบ non-recursive

1. ระบุขนาดของข้อมูลอินพุต (input size) และ basic operation ของอัลกอริทึม
   - Basic operation มักจะเป็นบรรทัดคำสั่งที่เรียก recursion

2. จากขนาดของอินพุต พิจารณาว่าอัลกอริทึมจำเป็นต้องวิเคราะห์กรณีเลวร้ายสุด (worst case) กรณีดีที่สุด (best case) และกรณีเฉลี่ย (average case)
   - จำนวนรอบการทำงานขึ้นอยู่กับลักษณะข้อมูลหรือไม่

3. เขียนฟังก์ชั่นเวลา T(n) ในรูปของ recurrence relation เพื่อนับรอบการทำงาน

4. ปรับฟังก์ชั่นเวลาของ basic operation ให้เข้าสู่ฟังก์ชั่นเวลาอ้างอิงที่ใกล้เคียงที่สุด

Recurrent relation

- สมมติให้ ลำดับเลขคณิต 1, 4, 7, 10, ... จงเขียนสมการทั่วไป (general equation) ของลำดับเลขคณิตเพื่อหาคำตอบในเทอมที่ n (a_n)

$$a_1 = 1$$

$$a_2 = 3 + a_1 = 3 + 1 = 4$$

$$a_3 = 3 + a_2 = 7$$

$$a_4 = 3 + a_3 = 10$$

ดังนั้น สมการทั่วไป $\mathbf{a_n = 3 + a_{n-1}}$

Recurrent relation in factorial

```
algorithm factorial(n):
    if n=0 then:
        return 1;
    else:
        return factorial(n-1)*n;
end algorithm
```

จำนวนรอบของ n-1

จำนวนรอบใหม่เพิ่มขึ้น

กำหนดให้ T(n) เป็นจำนวนรอบทำงานเมื่อขนาดข้อมูลเท่ากับ n ซึ่งมีค่าเท่ากับ จำนวนรอบของ T(n-1) รวมกับจำนวนรอบใหม่ที่เพิ่มขึ้น

$$T(n) = \begin{cases} T(n-1) + 1 & n > 0 \\ 0 & n = 0 \end{cases}$$

การแก้สมการ Recurrence relation

$$T(n) = \begin{cases} T(n-1) + 1 & n > 0 \\ 0 & n = 0 \end{cases}$$

1. Forward substitution แทนคำตอบโดยเริ่มต้นจากอินพุตที่เล็กที่สุดไปยังอินพุตที่ใหญ่สุด

T(1) = T(0) + 1 = 0 + 1 = 1

T(2) = T(1) + 1 = 1 + 1 = 2

......

T(n) = T(n-1) + 1 = (n-1) + 1 = n

2. backward substitution แทนคำตอบโดยเริ่มต้นจากอินพุตที่ใหญ่ที่สุดไปยังอินพุตที่เล็กสุด

T(n) = T(n-1) + 1

= (T(n-2) + 1) + 1 = T(n-2) + 2

= (T(n-3) +1) + 2 = T(n-3) + 3

....

= T(n-n) + n = T(0) + n = n

ตัวอย่าง

```
algorithm x(n):
    if n=1 then:
        return 1;
    else:
        return x(n/2)+1;
end algorithm
```

$$T(n) =$$