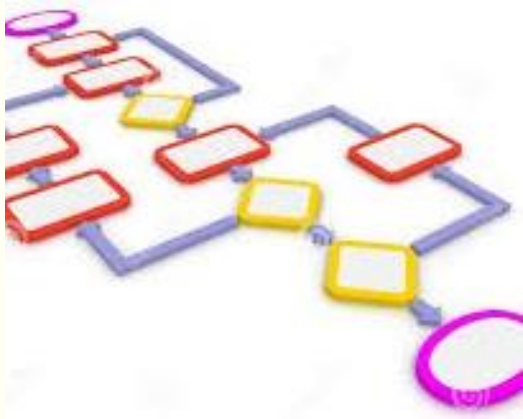
การออกแบบและวิเคราะห์ขั้นตอนวิธี

DESIGN AND ANALYSIS OF ALGORITHMS

02-212-212



อ.ธิดาวรรณ คล้ายศรี

บทที่1

Fundamental of Design and Analysis Algorithms

1.1 การแก้ปัญหาและขั้นตอนวิธี (Problems Solving and Algorithms)

1.2 An Introduction to Algorithm Design (บทนำการออกแบบขั้นตอนวิธี)

1.3 คณิตศาสตร์พื้นฐานเพื่อการวิเคราะห์ (Mathematic Preliminaries for Analysis)

1.4 Analysis of Algorithms (การวิเคราะห์อัลกอริธึ่ม)

1.5 Growth of Functions (แนวโน้มการเพิ่มขึ้นของฟังก์ชั่น)

1.2 บทนำการออกแบบขั้นตอนวิธี (An Introduction to Algorithm Design)

- **ขั้นตอนวิธี (Algorithm) = ?**

→วิธีการ/เครื่องมือที่ช่วยแก้ปัญหาอย่างเป็นขั้นตอนตามลำดับ เพื่อให้ได้ผลลัพธ์อย่างมีประสิทธิภาพ บางปัญหาต้องการวิธีการทางคอมพิวเตอร์ (computer algorithm) ซึ่งแตกต่างจากการแก้ปัญหาแบบสามัญสำนึก (heuristic)

→ขั้นตอนวิธีในการแก้ปัญหาอาจนำเสนอ: natural languages, pseudocode, flowcharts

เมื่อได้เขียนขั้นตอนวิธีแล้วก็จะทำการพัฒนาโปรแกรมด้วยภาษาคอมพิวเตอร์ (High-level programming languages: C++, Java) ให้ได้ผลลัพธ์ตามขั้นตอนวิธีที่ได้ออกแบบไว้

การวัดประสิทธิภาพของโปรแกรม

- ✓การใช้เนื้อที่ในหน่วยความจำ (Space/Memory)
  - ประสิทธิภาพของเวลาในการทำงาน (Time)
  - การวัดเวลาทำการเปรียบเทียบได้ลำบาก
  - พิจารณาอัตราการเติบโตของฟังก์ชัน (Growth Rates)

การวิเคราะห์ Space Complexity

✓ Maximum memory=? ที่รัน algorithm แล้วมีจริงๆ เท่าไหร่

- วิเคราะห์ว่าต้องใช้หน่วยความจำทั้งหมดเท่าไหร่ในการประมวลผลอัลกอริธึ่มนั้น รองรับจำนวนข้อมูลที่ส่งเข้ามาประมวลผล ได้มากที่สุดเท่าใด เพื่อให้อัลกอริธึ่มนั้นสามารถประมวลผลได้อยู่
- ทราบขนาดของหน่วยความจำที่จะต้องใช้ประมวลผลอัลกอริธึ่มโดยไม่กระทบการประมวลผลอื่นๆ
- เพื่อเลือกคุณลักษณะของคอมพิวเตอร์ที่จะใช้ติดตั้งโปรแกรมที่พัฒนาขึ้นได้อย่างเหมาะสม

องค์ประกอบของ Space Complexity

✓**Instruction Space**

-จำนวนของหน่วยความจำที่คอมไพเลอร์จำเป็นต้องใช้ขณะทำการคอมไพล์โปรแกรม

✓**Data Space**

-จำนวนหน่วยความจำที่ต้องใช้สำหรับเก็บค่าคงที่ และตัวแปรทั้งหมดที่ต้องใช้ในการประมวลผลโปรแกรม

- **Static memory allocation**

จำนวนของหน่วยความจำที่ต้องใช้อย่างแน่นอน ไม่มีการเปลี่ยนแปลง ประกอบด้วยหน่วยความจำที่ใช้เก็บค่าคงที่และตัวแปรประเภท array เช่น การประกาศตัวแปร

```
int    a, b;
char   s[10],   c;
```

องค์ประกอบของ Space Complexity

- **Dynamic memory allocation**

  จำนวนของหน่วยความจำที่ใช้ในการประมวลผลสามารถเปลี่ยนแปลงได้ และจะทราบจำนวนหน่วยความจำที่จะใช้ก็ต่อเมื่อโปรแกรมกำลังทำงานอยู่ เช่น การใช้ pointer และมีการจองเนื้อที่ให้หน่วยความจำด้วยคำสั่ง malloc();

```
int  *p;
p = malloc(sizeof(int)*2);
```

✓**Environment Stack Space**

-จำนวนหน่วยความจำที่ต้องใช้ในการเก็บผลลัพธ์ของข้อมูลเอาไว้ เพื่อรอเวลาที่จะนำผลลัพธ์นั้นกลับไปประมวลผลอีกครั้ง (พบใน recursive function)

การวิเคราะห์ Time Complexity

คือ เวลาที่เครื่องคอมพิวเตอร์ต้องใช้ในการประมวลผลอัลกอริธึ่ม ซึ่งวิเคราะห์เพื่อ:

- ประมาณการเวลาทั้งหมดที่ต้องใช้ในโปรแกรมได้
- มุ่งแก้ไขไปที่อัลกอริธึ่มที่ใช้เวลาในการประมวลผลนานๆ ทำให้ไม่ต้องแก้ไขทั้งโปรแกรม
- เลือกคุณลักษณะของคอมพิวเตอร์ที่จะใช้ติดตั้งโปรแกรมที่พัฒนาขึ้นได้อย่างเหมาะสม

เวลาในการประมวลผลของโปรแกรม

- Compile Time คือ เวลาที่ใช้ในการตรวจสอบไวยากรณ์ (syntax) ของ code ว่าเขียนได้ถูกต้องหรือไม่
- Run Time หรือ Execution Time คือ เวลาที่เครื่องทำการรันโปรแกรม
- คอมพิวเตอร์ใช้ในการประมวลผลลัพธ์

ประสิทธิภาพของอัลกอริธึ่ม

พิจารณาจำนวนอิลิเมนต์ (Elements) ที่จะประมวลผลหรือจากจำนวนรอบการทำงานของตัวดำเนินการนั้นๆ

- f(n) = efficiency

→ อัตราการเติบโตของฟังก์ชั่น (growth rates) ที่บอกความสัมพันธ์ระหว่างจำนวนข้อมูลนำเข้ากับความเร็วในการประมวลผล

- พิจารณาจากส่วนการทำงานของวนรอบประมวลผล (loop) เป็นสำคัญ

ลูปแบบเชิงเส้น (Linear Loops)

= มีการเพิ่ม หรือลดค่าภายในลูปแบบคงที่

```
for ( i = 0; i < 1000; i++ )
    application code
```

ประสิทธิภาพของอัลกอริธึ่มคือ f(n) = n

```
for ( i = 0; i < 1000; i+= 2 )
    application code
```

ประสิทธิภาพของอัลกอริธึ่ม f(n) = n / 2

→เมื่อพล๊อตจุดลงกราฟจะมีลักษณะเป็นเส้นตรงเหมือนกัน

ลูปแบบลอการิธมิค (Logarithmic Loops)

= เพิ่มหรือลดค่าภายในลูปสองเท่า

- Multiply Loops

```
for( i =1; i <= 1000; i* = 2 )
        application code
```

- Divide Loops

```
for( i =1000; i  >= 1; i / = 2 )
        application code
```

ประสิทธิภาพของอัลกอริทึม $f(n) = [\log_n]$ หรือ $[\log_2 n]$

ลูปแบบซ้อน (Nested Loops)

= ภายในลูป จะมีลูปซ้อนอีกลูปหนึ่ง

✓ ยอดรวมที่ได้คือผลคูณของจำนวนลูปชั้นใน (Inner Loop) กับจำนวนของลูปชั้นนอก (Outer Loop)

✓ ลูปแบบซ้อนชนิดลอการิธมิคเชิงเส้น (Linear Logarithmic)

- ลูปแบบซ้อนชนิดกำลังสอง (Quadratic)
- ลูปแบบซ้อนกำลังสองชนิดขึ้นต่อกัน (Dependent Quadratic)

ลูปแบบซ้อนชนิดลอการิธมิคเชิงเส้น

```
for (i = 0; i < 10; i++)
    for (j = 1; j <= 10; j*= 2)
        application code
```

จำนวนรอบคือ 10 $\log_{10}$

f(n) = n log $_{n}$

ลูปแบบซ้อนชนิดกำลังสอง (Quadratic)

```
for (i = 0; i < 10; i++)
    for (j = 0; j < 10; j++)
        application code
```

จำนวนรอบคือ 10 * 10

$$f(n) = n^2$$

ลูปแบบซ้อนชนิดขึ้นต่อกัน (Dependent Quadratic)

```
for (i = 0; i < 10; i++)
    for (j = 0; j <= i ; j++)
        application code
```

จำนวนรอบการทำงานในลูปในคือ 1+2+3 ... + 10 = 55

f จำนวนรอบเฉลี่ยคือ 55/10 = 5.5

เทียบได้เท่ากับ **(n + 1)/2**

(n) = n * (n+1)/2