Cấu trúc dữ liệu và thuật toán

2025 Tuần 1

Giảng viên: Nguyễn Thị Tâm - Ngô Thế Quyền

§ Độ phức tạp thuật toán §

Phần 1: Kiến thức nhắc lại

- Định nghĩa các ký hiệu O, Ω, Θ và hiểu ý nghĩa của chúng trong việc đánh giá độ phức tạp thuật toán.
- Định lý "Master Theorem" (Định lý chính/Định lý thợ) để xác định độ phức tạp cho các thuật toán chia để trị
  - Dùng để giải các công thức đệ quy dạng $T(n) = aT(\frac{n}{b}) + \Theta(n^k log^p n)$, với $a \geq 1, b > 1, k \geq 0$
  - Bài toán ban đầu được chia thành a bài toán con có kích thước mỗi bài toán con là $\frac{n}{b}$ chi phí để tổng hợp các bài toán con là $\Theta(n^k log^p n)$
    1. Nếu $a > b^k$, $T(n) = \Theta(n^{log_b a})$
    2. Nếu $a = b^k$
       * Nếu $p > -1$, $T(n) = \Theta(n^{log_b a} log^{p+1} n)$
       * Nếu $p = -1$, $T(n) = \Theta(n^{log_b a} loglogn)$
       * Nếu $p < -1$, $T(n) = \Theta(n^{log_b a})$
    3. Nếu $a < b^k$
       * Nếu $p \geq 0$, $T(n) = \Theta(n^k log^p n)$
       * Nếu $p < 0$, $T(n) = O(n^k)$
- Định lý chính rút gọn Giả sử $a \geq 1, b > 1, c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ quy:

$$T(n) = aT(\tfrac{n}{b}) + cn^k$$

  Xác định với $n \geq 0$

  1. nếu $a > b^k$ thì $T(n) = \Theta(n^{log_b^a})$
  2. nếu $a = b^k$ thì $T(n) = \Theta(n^k log(n))$
  3. nếu $a < b^k$ thì $T(n) = \Theta(n^k)$
- Định lý chính cho cho các thuật toán chia để trị (kiểu quan hệ trừ dần)

$$T(n) = \begin{cases} c & \text{nếu } n \leq 1 \\ aT(n-b) + f(n) & \text{nếu } n > 1 \end{cases}$$

với $c, a > 0, b > 0, k \leq 0, f(n) = O(n^k)$, ta có:

$$T(n) = \begin{cases} O(n^k) & \text{nếu } a \leq 1 \\ O(n^{k+1}) & \text{nếu } n = 1 \\ O(n^k a^{\frac{n}{b}}) & \text{nếu } a > 1 \end{cases}$$

Phần 2: Bài tập

Bài 1 Xác định độ phức tạp

1. $$T(n) = \begin{cases} 3T(n-1) & \text{nếu } n \leq 0 \\ 1 & \text{ngược lại} \end{cases}$$

2. $$T(n) = \begin{cases} 2T(n-1) - 1 & \text{nếu } n \leq 0 \\ 1 & \text{ngược lại} \end{cases}$$

Bài 2 Xác định độ phức tạp thuật toán của các đoạn chương trình sau:

```java
public static int count(int[] a)
{ // Count triples that sum to 0.
    int n = a.length;
    int cnt = 0;
    for (int i = 0; i < n; i++)
        for (int j = i+1; j < n; j++)
            for (int k = j+1; k < n; k++)
                if (a[i] + a[j] + a[k] == 0)
                    cnt++;
    return cnt;
}
```

```java
public void function(int n) {
    int i, j, k , count =0;
    for(i=n/2; i<=n; i++)
        for(j=1; j + n/2<=n; j++)
            for(k=1; k<=n; k= k * 2)
                count++;
}
```

```java
public void function(n) {
    for(int i = 1 ; i <= n ; i++)
        for(int j = 1 ; j <= n ; j += i)
            System.out.println‘“’(*) ;
}
```

```java
public void function(int n) {
    int i=1;
    while (i < n) {
        int j=n;
        while(j > 0)
            j = j/2;
        i=2*i;
    }
}
```

Bài 3 Chứng minh độ phức tạp của $\sum_{i=1}^{n} logi$ là $O(nlogn)$

Bài 4 Xác định độ phức tạp của chương trình đệ quy

```java
public void function(int n) {
    if(n <= 1) return;
        for (int i=1 ; i <= 3 ; i++ )
            function(n-1);
```

```
}
```

```
 public void function (int n) {
    if(n <= 1) return;
    for(int i = 1; i < n; i++)
        System.out.println("*");
    function (0.8n) ;
}
```

```
 public void function (int n) {
    if(n < 2) return;
    for(int i = 1; i < 8; i++)
        function(n/2);
    for (int i = 1; i <= n*n*n; i++)
        System.out.println("*");
}
```

```
public void function(int n) {
    if(n <= 1) return;
    if(n > 1) {
        System.out.println("*");
        function(n/2);
        function(n/2);
    }
}
```