# Phân tích độ phức tạp thuật toán đệ qui

Nhóm 2-CS112.L23.KHCL

Trần Hồ Thiên Phước

Lê Văn Trí

Phạm Nguyễn Công Danh

- Có 2 loại chương trình đệ quy
  - Trực tiếp (Tính giai thừa, tháp Hà Nội, Fibonacci,…)
  - Gián tiếp (xét tính chẵn lẻ,…)

Chúng ta cũng có thể chia ra: đệ quy tuyến tính, đệ quy đuôi, đệ quy nhị phân, đệ quy đa tuyến, đệ quy lồng, đệ quy tương hỗ.

- Làm sao để tính độ phức tạp chương trình đệ quy ?
  - Thành lập phương trình đệ quy T(n)
  - Giải phương trình đệ quy bằng:
    - Phương pháp truy hồi
    - Phương pháp đoán nghiệm
    - Áp dụng công thức đối với phương trình đệ quy đã có lời giải

# VÍ DỤ VỀ GIẢI PT ĐỆ QUY BẰNG PHƯƠNG PHÁP TRUY HỒI

# n! = n*(n-1)*(n-2)*.....*2*1

$$n! = \begin{cases} 1 \text{ nếu } n = 0 \\ n(n-1)! \end{cases}$$

```python
def Factorial(n):
    if n == 0:
        return 1
    return n * Factorial(n - 1)
```

- Gọi T(n) là thời gian tính n!
- Thì T(n-1) là thời gian tính (n-1)!
- Khi n = 0 thì CT return 1 tốn O(1), do đó ta có T(0) = 1
- Khi n > 0 thì CT phải :
  - Tính (n-1)!, tốn thời gian T(n-1)
  - Tính n*(n-1)! và return kết quả tốn hằng thời gian, cho là 1

$$\mathbf{T(n)} \begin{cases} \mathbf{1} & \textbf{\textit{nếu}}\ \boldsymbol{n=0} \\ \boldsymbol{T(n-1)+1} & \textbf{\textit{nếu}}\ \boldsymbol{n>0} \end{cases}$$

# VÍ DỤ 1 VỀ GIẢI PT ĐỆ QUY BẰNG PHƯƠNG PHÁP TRUY HỒI

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

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

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

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

……

T(n) = T(n-i) + I

- Quá Trình nên kết thúc khi n – i = 0 ⇔ I = n.
- Khi đó ta có T(n) = T(0) + n = 1+n = O(n)

# Lời giải tổng quát

- Giải thuật chia để trị:
  - Phân rã bài toán lớn thành các bài toán con
    - Một bài toán lớn có kích thước n, thành a bài toán con có kích thước n/b
  - Tổng hợp các lời giải của các bài toán con để có được lời giải của bài toán lớn
    - Thời gian tổng hợp a bài toán con tốn d(n) thời gian
- Phương trình đệ quy cho giải thuật trên:
  - $T(1)=1$
  - $T(n)=aT(n/b)+d(n)$

# Lời giải tổng quát

- Áp dụng phương pháp truy hồi:

$$
\begin{aligned}
T(n) &= aT(n/b) + d(n) \\
&= a[T(n/b/b) + d(n/b)] + d(n) \\
&= a^2T(n/b^2) + ad(n/b) + d(n) \\
&= a^2[aT(n/b^3) + d(n/b^2)] + ad(n/b) + d(n) \\
&= a^3T(n/b^3) + a^2d(n/b^2) + ad(n/b) + d(n) \\
&= \ldots \\
&= a^kT(n/b^k) + \sum a^i d(n/b^i)
\end{aligned}
$$

# Lời giải tổng quát

- Áp dụng phương pháp truy hồi:

$$T(n) = aT(n/b) + d(n)$$

$$= a^k T(n/b^k) + \sum a^i d(n/b^i)$$

– Quá trình kết thúc khi $n/b^k = 1$ hay $k = \log_b n$

$$T(n) = a^k + \sum a^i d(n/b^i)$$

# Lời giải tổng quát

- Nghiệm thuần nhất (homogeneous solutions ):

  $n^{log_b a}$

- d(n): hàm tiến triển (driving function)
- Nghiệm chính xác sẽ là nghiệm chính xác nếu d(n) = 0, với mọi n
- Nếu d(n) > 0, ta có nghiệm riêng:

$$\sum_{i=0}^{k-1} a^i \mathrm{d}\left(\frac{n}{b^i}\right) = \sum_{i=0}^{k-1} a^i \mathrm{d}\left(\frac{b^k}{b^i}\right) = \sum_{i=0}^{k-1} a^i \mathrm{d}\left(b^{k-i}\right)$$

## Lời giải tổng quát

- Nếu nghiệm thuần nhất lớn nghiệm riêng thì độ phức tạp là nghiệm thuần nhất
- Nếu nghiệm riêng lớn hơn nghiệm thuần nhất thì độ phức tạp là nghiệm riêng
- Tuy nhiên, tính nghiệm không phải lúc nào cũng dễ!

# Lời giải tổng quát

- Ta sẽ tính nghiệm riêng trong trường hợp d(n) có dạng đặc biệt
- Hàm nhân, hàm có tính chất nhân (multiplicative function):
  - Hàm d(n) có tính nhân nếu và chỉ nếu d(x.y) = d(x).d(y)
  - Ví dụ:
    - d(n) = n2 là hàm nhân vì d(x.y) = (x.y)2 = x2 .y2 = d(x).d(y)
    - d(n) = 3n2 không phải là hàm nhân

## Lời giải tổng quát

- Nếu d(n) là hàm nhân, ta có nghiệm riêng:

$$\sum_{i=0}^{k-1} a^i \mathrm{d}\left(b^{k-i}\right) = \sum_{i=0}^{k-1} a^i \left[d(b)\right]^{k-i}$$

$$= \left[d(b)\right]^k \sum_{i=0}^{k-1} \left[\frac{a}{d(b)}\right]^i$$

$$= \frac{a^k - \left[d(b)\right]^k}{\frac{a}{d(b)} - 1}$$

## Lời giải tổng quát

- Nếu a > d(b), $a^k > [d(b)]^k$

$$T(n)=O(a^k) = O(a^{log_b^n})=O(n^{log_b^a})$$

- Nếu a < d(b)

$$T(n)=O(d(b)^k) = O\left(d(b)^{log_b^n}\right) = O(n^{log_b^{d(b)}})$$

- Nếu a = d(b)

$$\sum_{i=0}^{k-1} a^i d\left(b^{k-i}\right) = \sum_{i=0}^{k-1} a^i [d(b)]^{k-i} = [d(b)]^k \sum_{i=0}^{k-1} \left[\frac{a}{d(b)}\right]^i$$

$$= d(b)^k k = a^k k$$

$$T(n) = O(n^{log_b^a} log\, n)$$

## Lời giải tổng quát

- TH d(b) không phải hàm nhân?

**-> Tính trực tiếp nghiệm riêng và nghiệm thuần nhất.**


WHEN? WHY? WHEN? WHY?
WHAT? HOW? WHY?
WHEN?
WHO?
WHAT?
WHERE?
WHEN?
HOW? HOW?
WHERE?
WHICH?
HOW?
WHEN? WHERE?
HOW?
WHAT? HOW? WHY?
WHY? HOW?
WHAT? HOW? WHY?
WHO? WHERE? WHAT? HOW?
WHAT?
WHY? HOW? WHERE?
WHO? WHOSE?
WHERE? WHAT? HOW?
WHO?
WHERE? WHICH? WHOSE? WHEN? WHY?
WHERE?
HOW? WHO?
WHO? WHERE? WHAT? HOW?
WHAT? HOW? WHY?
HOW?

```
def mergeSort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]
        mergeSort(L)
        mergeSort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

```

Mergesort(A):

```
        n ← length(A)
        if ( n<2 ) return
        mid ← n/2
        left ← array of size (mid)
        right ← array of size (n-mid)
        for i ←0 to mid – 1
                left[i] ←A[i]
        for i ←mid to n - 1
                right[i] ←A[i]
        Mergesort(left)
        Mergesort(right)
        Merge(left, right, A)
```

$$T(n)=\begin{cases} c & nếu \;\; n=1 \\ 2T(n/2)+c'n & nếu \;\; n>1 \end{cases}$$