Chương 1. Cấu trúc dữ liệu và giải thuật

Ths. Phạm Thanh An

Khoa Công nghệ thông tin

Trường Đại học Ngân hàng TP.HCM

LOGO

Nội dung

- Giải thuật và cấu trúc dữ liệu
  - Giải thuật và các đặc trưng của giải thuật
  - Diễn đạt giải thuật
  - Kiểu dữ liệu, ADT, Cấu trúc dữ liệu
- Phân tích và thiết kế giải thuật
  - Thiết kế giải thuật
  - Phân tích giải thuật
- Một số lớp các giải thuật

Mục tiêu

- Tìm hiểu các nội dung:
  - Thiết kế và phân tích được giải thuật
  - Hiểu rõ về Kiểu dữ liệu, Kiểu dữ liệu trừu tượng, Cấu trúc dữ liệu.
  - Đánh giá độ phức tạp của giải thuật cơ bản

Giải bài toán bằng máy tính

- Giải quyết một bài toán:
  - Làm gì ?
  - Làm như thế nào ?
- Giải quyết Bài toán Tin học $\Rightarrow$ phải:
  - Tổ chức biểu diễn các đối tượng thực tế
  - Xây dựng trình tự các thao tác xử lý trên các đối tượng dữ liệu đó

Giải bài toán bằng máy tính

- Hai yếu tố tạo nên một chương trình máy tính
  - Cấu trúc dữ liệu
  - Giải thuật

Cấu trúc dữ liệu + Giải thuật = Chương trình

Giải thuật

- **Định nghĩa**: là dãy các câu lệnh chặt chẽ và rõ ràng xác định một trình tự các thao tác trên một số đối tượng nào đó, sao cho sau một số hữu hạn bước thực hiện ta đạt được kết quả mong muốn
- Mỗi thuật toán có một **dữ liệu vào (Input)** và một **dữ liệu ra (Output)**;

Giải thuật

- Lý thuyết giải thuật quan tâm đến những vấn đề sau :
  - 1. Giải được bằng giải thuật :
  - 2. Tối ưu hóa giải thuật :
  - 3. Triển khai giải thuật:

Đặc trưng của giải thuật

- Tính xác định :
- Tính dừng (hữu hạn):
- Tính đúng đắn:
- Tính phổ dụng:
- Tính khả thi:

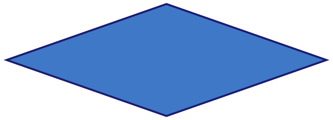
Diễn đạt giải thuật

- Dạng lưu đồ (sơ đồ khối)
- Dạng ngôn ngữ tự nhiên (Ngôn ngữ liệt kê từng bước)
- Dạng mã giả
- Ngôn ngữ lập trình

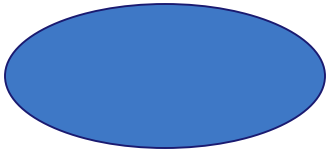
Diễn đạt giải thuật

Các nút biểu diễn giải thuật bằng sơ đồ khối

Nút thao tác:



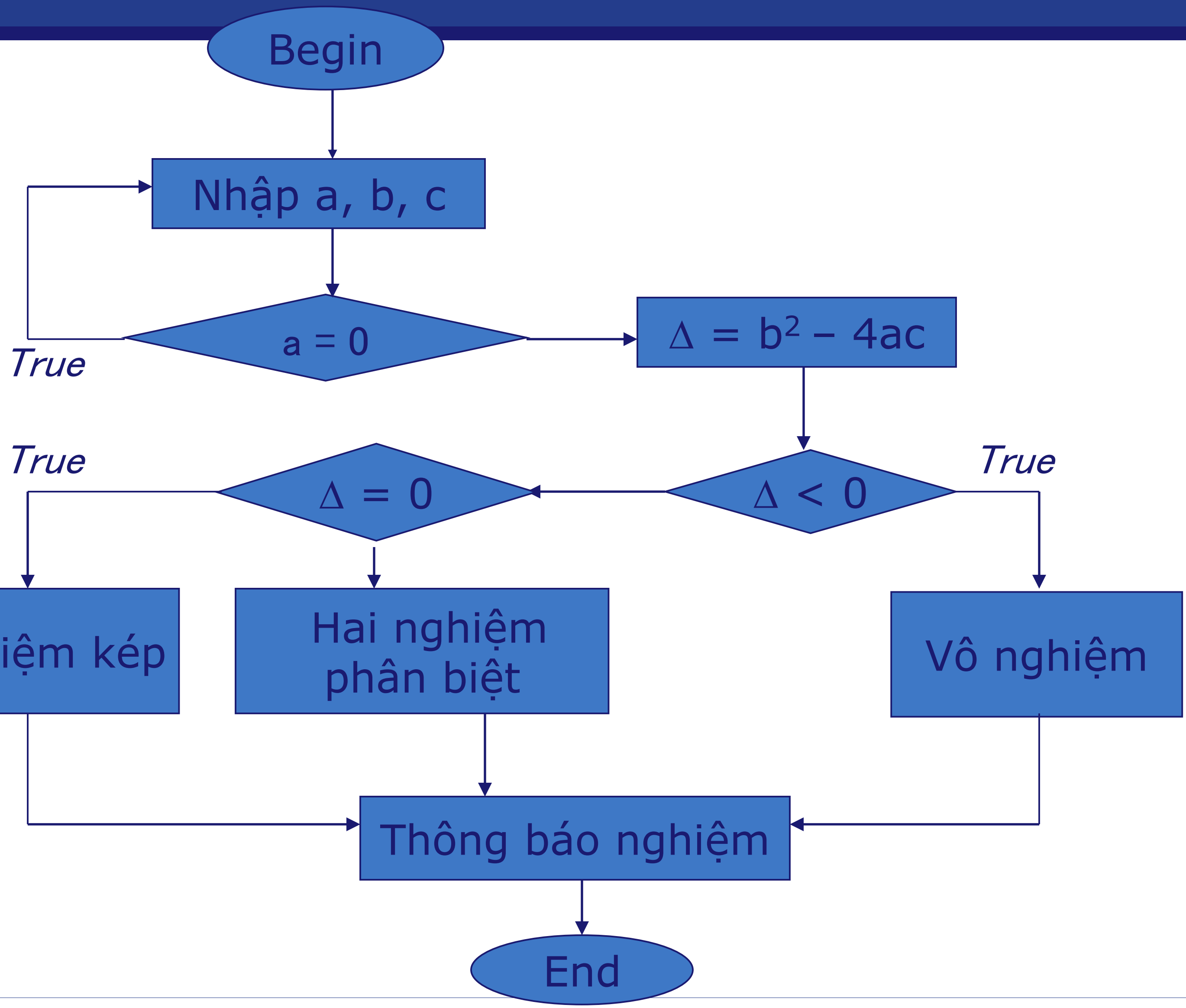
Nút điều khiển: trong đó ghi điều kiện cần kiểm tra trong quá trình tính toán.



Nút khởi đầu ,kết thúc:

Cung :

Ví dụ : Giải PT: $ax^2 + bx + c = 0$, giải thuật mô tả bằng sơ đồ khối



Diễn đạt giải thuật

- Ví dụ 1: Giải thuật xác định n là số nguyên tố
  - Bước 1: Ghi nhận n
  - Bước 2: Nếu $n \le 1 \rightarrow$ n ko nguyên tố $\rightarrow$ dừng
  - Bước 3: Nếu $n > 2$, gán $i \leftarrow 2$
  - Bước 4: Nếu $i \ge \sqrt{n}$ hay n chia hết cho i $\rightarrow$ bước 6
  - Bước 5: Gán $i \leftarrow i+1$, trở lại bước 4
  - Bước 6:
    - Nếu $i > \sqrt{n} \rightarrow$ n nguyên tố $\rightarrow$ dừng
    - Ngược lại, n không là nguyên tố $\rightarrow$ dừng

Diễn đạt giải thuật (tt)

- Ví dụ 2: Giải thuật tìm phần tử thứ n của dãy số Fibonacci
  - Bước 1: Ghi nhận n
  - Bước 2: Nếu n=1 hay n=2 → u_n=1 → dừng
  - Bước 3: Nếu n > 2, gán a←1, b←1, i←1
  - Bước 4: Gán c←a+b, a←b, b←c
  - Bước 5:
    - Nếu i = n - 2 → u_n=c → dừng
    - Ngược lại i ← i+1, quay lại bước 4

Diễn đạt giải thuật (tt)

- Ví dụ 3: tìm phần tử lớn nhất trong mảng A
  - Giải thuật timMax(A, n)

    Input: Mảng A, *gồm* n số nguyên

    Output: Giá trị lớn nhất của A

    $Max \leftarrow A[0]$

    for $i \leftarrow 1$ to $n - 1$ do

        if $A[i] > Max$ then

            $Max \leftarrow A[i]$

    return Max

Kiểu dữ liệu, Kiểu dữ liệu trừu tượng

- Kiểu dữ liệu (Data type)
- Kiểu dữ liệu trừu tượng (ADT - abstract data type):
  - Một kiểu dữ liệu trừu tượng là một mô hình toán học cùng với một tập hợp các phép toán (operation) được định nghĩa trên mô hình đó.

Cấu trúc dữ liệu

- Cấu trúc dữ liệu (Data structure)
- Trong ngôn ngữ lập trình, có một số cấu trúc dữ liệu riêng của nó được gọi là CTDL tiền định.

Cấu trúc lưu trữ (trong/ngoài)

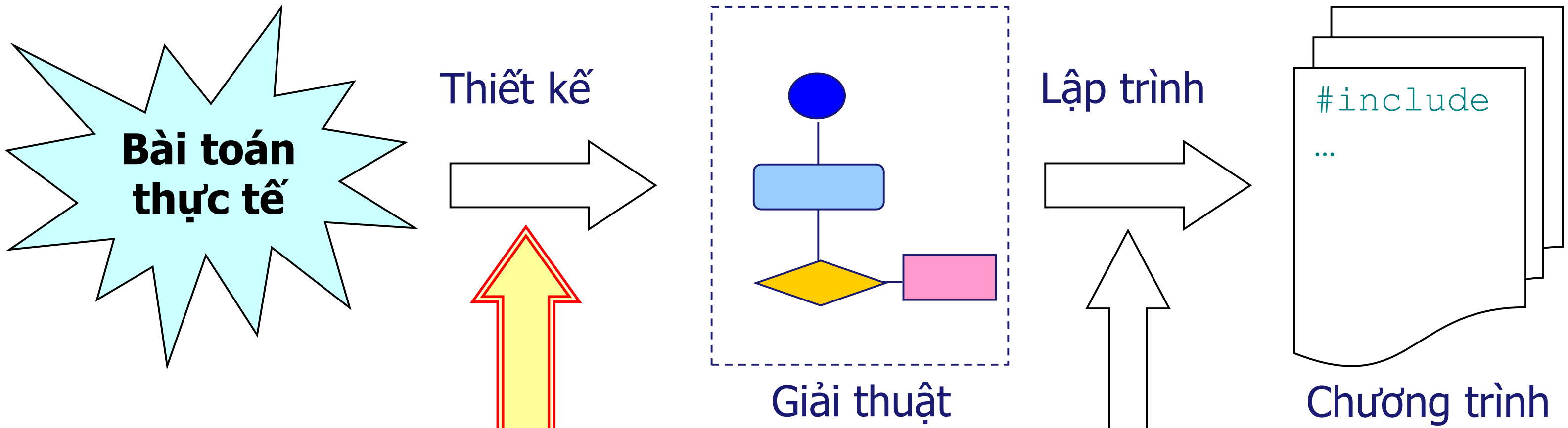
- Là các biểu diễn cấu trúc dữ liệu trên bộ nhớ (trong/ngoài) của máy tính
- Có nhiều cấu trúc lưu trữ khác nhau cho cùng một cấu trúc dữ liệu

Mối quan hệ giữa Giải thuật và Cấu trúc dữ liệu

- Đối tượng xử lý của giải thuật chính là dữ liệu
- Với một cấu trúc dữ liệu, sẽ có những giải thuật tương ứng.
- Khi cấu trúc dữ liệu thay đổi thường giải thuật cũng phải thay đổi theo.

Thiết kế giải thuật

❖ Từ bài toán đến chương trình



Kỹ thuật thiết kế giải thuật:

Chia để trị, quy hoạch động, backtracking ..vv

- **Ngôn ngữ lập trình:**
- **PASCAL, C/C++, JAVA, C#**

Thiết kế giải thuật (tt)

- Với một vấn đề đặt ra, làm thế nào để đưa ra thuật toán giải quyết nó?
- Chiến lược thiết kế:
  - Chia-để-trị (divide-and-conquer)
  - Quy hoạch động (dynamic programming)
  - Quay lui (backtracking)
  - Tham lam (greedy method)

Thiết kế giải thuật (tt)

- Module hoá và việc giải quyết bài toán
  - Chiến thuật chia để trị (divide-conquer):
  - Để thực hiện chiến thuật này, thường có hai cách thiết kế:

1. Từ trên xuống (Top-Down Design).
2. Tinh chỉnh từng bước

Thiết kế giải thuật (tt)

- **Sau đây là lược đồ của kỹ thuật chia-để-trị:**

```
DivideConquer (A,x)    // tìm nghiệm x của bài toán A.
{
  if (A đủ nhỏ)
        Solve (A);
   else {
        Chia bài toán A thành các bài toán con
                A1, A2,…, Am;
        for (i = 1; i <= m ; i ++)
            DivideConquer (Ai , xi);
   Kết hợp các nghiệm xi của các bài toán con Ai (i=1, …, m)
   để nhận được nghiệm x của bài toán A;
   }
}
```

Thiết kế giải thuật (tt)

- Tinh chỉnh từng bước:
  - Biểu diễn ý tưởng bằng ngôn ngữ tự nhiên
  - Cụ thể từng phần, thay đổi bằng ngôn ngữ chương trình
  - Cuối cùng ta có chương trình

Thiết kế giải thuật (tt)

- Ví dụ: Bài toán sắp xếp một dãy n số, theo thư tự tăng dần
  - Chọn số bé nhất trong n số để vào vị trí thứ 1
  - Chọn số bé nhất trong n-1 số còn lại để vào vị trí thứ 2
  - …………………
  - Chọn số bé nhất trong 2 số còn lại để vào vị trí thứ n-1

Thiết kế giải thuật (tt)

- ➔ *For (i= 1, i <= n-1, i++)*

  {- Chọn số bé nhất trong các số $x_i, x_{i+1}, \ldots, x_n$

  - Đổi chỗ cho xi

  }

- ➔ *For (i= 1, i <= n-1, i++)*

  {- tg = x[i]

  -So sánh tg với các số từ x_{i+1} *->* x_n *. Nếu x[i] > các số đó thì lại lấy số đó làm số tg*

  - đổi chỗ x[i] và tg

  }

Thiết kế giải thuật (tt)

```
for (i= 1; i <= n-1; i++)
 for(j = i+1; j <=n; j++) {
  If  (x[j]  <  x[i])
  {
     Tg = x[i] ;
      X[i] = x[j];
      X[j] := tg ;
   }
}
```

Thiết kế giải thuật (tt)

- Ví dụ 3: Tìm tất cả các số tự nhiên có hai chữ số, khi đảo trật tự của hai số đó sẽ tạo được một số nguyên tố cùng nhau với số đã cho
  - Phân tích giả thiết
    - Gọi x=ab là số có hai chữ số cần tìm
    - a,b = 0…9
    - a > 0
    - (ab,ba)=1

Thiết kế giải thuật (tt)

- Ví dụ 3 (tt)
  - Tinh chỉnh 1
    - x = 10…99
    - x’ = 10*donvi(x)+chuc(x)
    - (x,x’)=1 ⇔ USCLN(x,x’)=1
  - Tinh chỉnh 2
    - x chạy từ 10 đến 99
    - y=10*donvi(x)+chuc(x)
    - nếu USCLN(x,y)=1 thì Xuất(x)

Phân tích Giải thuật (tt)

- Khi một giải thuật được xây dựng, hàng loạt yêu cầu đặt ra
  - Yêu cầu về tính đúng đắn của giải thuật
  - Tính đơn giản của giải thuật.
  - Yêu cầu về không gian :
  - Yêu cầu về thời gian :

Phân tích Giải thuật (tt)

- Giải quyết bài toán
  - Phải đứng trước việc lựa chọn giải thuật nào ?
  - Dựa trên cơ sở nào để lựa chọn ?
- Có hai mục tiêu trái ngược
  - Thuật toán dễ hiểu, cài đặt và gỡ lỗi (1).
  - Thuật toán sử dụng hiệu quả tài nguyên máy tính, đặc biệt chạy càng nhanh càng tốt (2).

Phân tích Giải thuật (tt)

- Độ phức tạp không gian (Space complexity)
  - Dung lượng bộ nhớ mà thuật toán đòi hỏi
- Độ phức tạp thời gian (Time complexity)
  - Thời gian thực hiện thuật toán

Phân tích thời gian thực hiện giải thuật

- Thời gian thực hiện giải thuật phụ thuộc vào các yếu tố sau:
  - Dữ liệu vào
  - Tốc độ thực hiện các phép toán của máy tính (phần cứng máy tính)
  - Trình biên dịch

Phân tích thời gian thực hiện giải thuật

- Sử dụng các công cụ toán học để đánh giá thời gian chạy của giải thuật:
- Gọi n là kích thước của dữ liệu vào, thời gian thực hiện của giải thuật có thể biểu diễn là một như hàm của n: hàm T(n)

Tiến trình phân tích thời gian thực hiện giải thuật

- Bước 1: Phân tích kích thước dữ liệu vào
- Bước 2: Phân tích (toán học) tìm ra giá trị trung bình, và giá trị xấu nhất cho mỗi đại lượng cơ bản.

Độ phức tạp tính toán của giải thuật

- Ví dụ 4:
  - Giải thuật A, độ phức tạp thời gian $T_a(n)$
  - Giải thuật B, độ phức tạp thời gian $T_b(n)$
  - Khi n lớn, $T_a(n) >> T_b(n)$. Có thể kết luận giải thuật A chậm hơn giải thuật B.

Ký pháp để đánh giá độ phức tạp tính toán của giải thuật

- Ký hiệu O (big-Oh): hàm f(n) và g(n), ta nói:

  f(n) = O(g(n)), nếu tồn tại các hằng số dương c và n_o sao cho $f(n) \le cg(n)$ khi $n \ge n_o$.

- Ký hiệu này dùng để chỉ chặn trên của một hàm
- Ý nghĩa: Tốc độ tăng của hàm f(n) không lớn hơn hàm g(n)

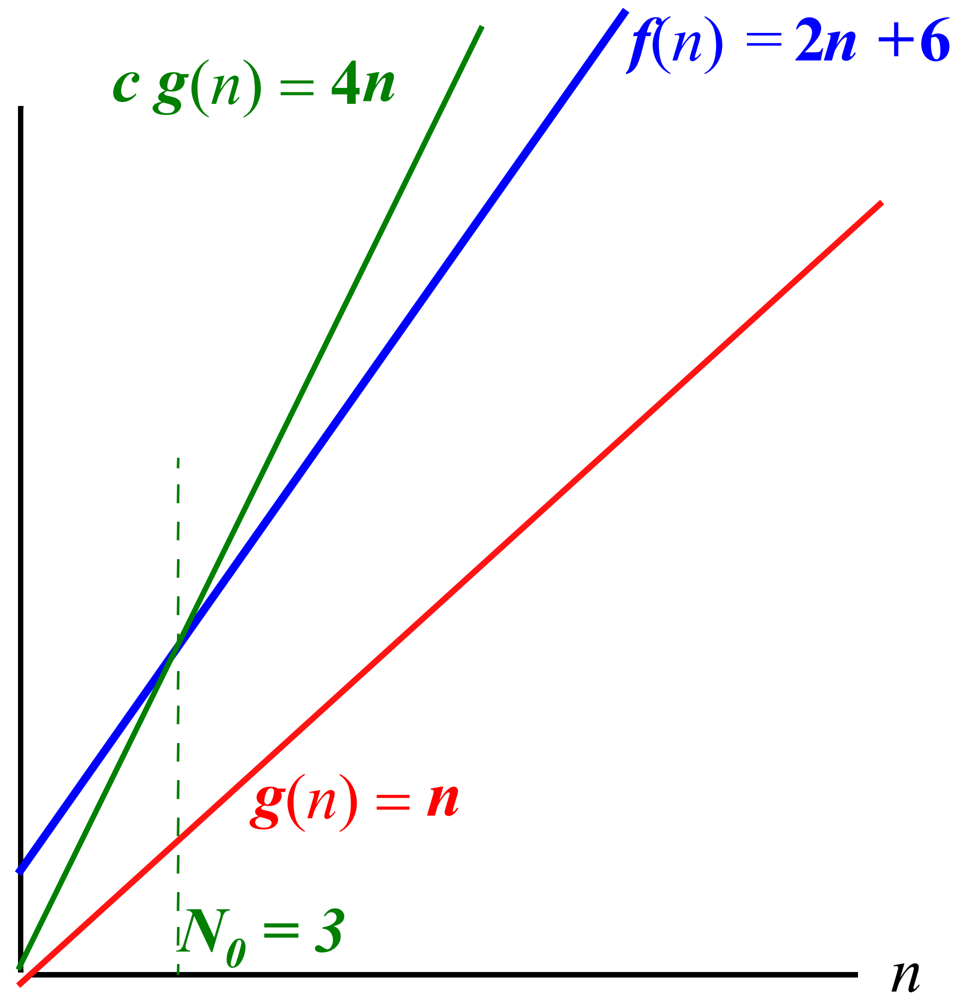
Ký pháp để đánh giá độ phức tạp tính toán của giải thuật

- Ví dụ :

f(n) = 2n+6,

g(n) = n và c = 4 , n_0=3

- f(n)= O(n)



Ký pháp để đánh giá độ phức tạp tính toán của giải thuật

- Định nghĩa Ω:

  - $f(n) = \Omega(g(n))$, nếu tồn tại các *hằng số dương c* và n_o sao cho $f(n) \geq cg(n)$ khi $n \geq n_o$
  - $f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$

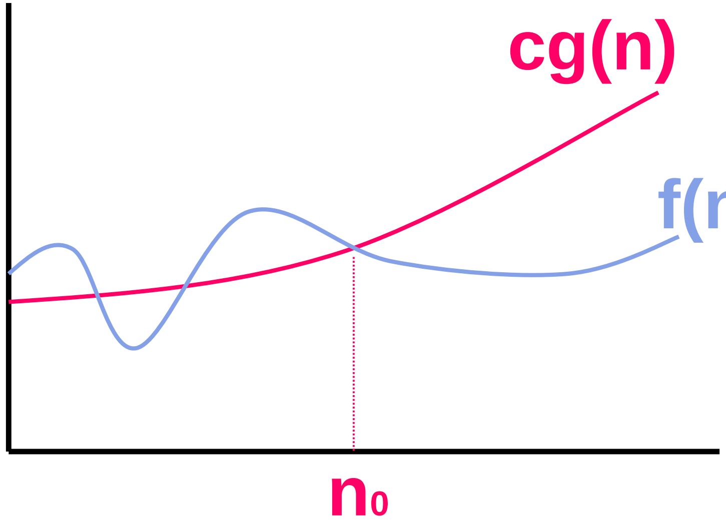
- Định nghĩa Θ

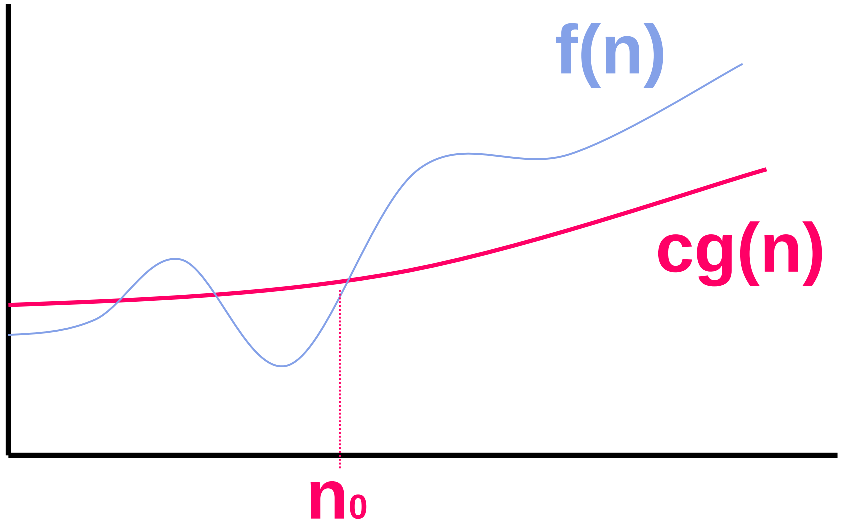
  $f(n) = \Theta(g(n))$, nếu tồn tại các hằng số dương c_1, c_2 và n_0 sao cho $c_1.g(n) \leq f(n) \leq c_2.\ g(n)$ với mọi $n > n_0$

Ký pháp để đánh giá độ phức tạp tính toán của giải thuật

$$f(n)=O(g(n))$$

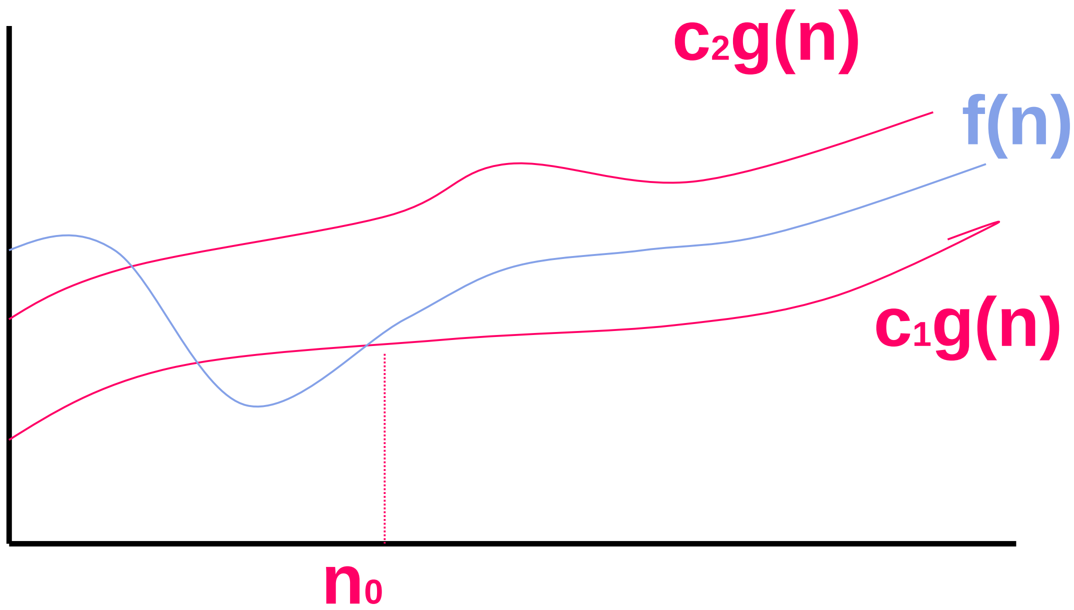
$$f(n)=\Omega(g(n))$$





Ký pháp để đánh giá độ phức tạp tính toán của giải thuật

$$f(n) = \Theta(g(n))$$



Ký pháp để đánh giá độ phức tạp tính toán của giải thuật

- Ta nói độ phức tạp tính toán của giải thuật có cấp f(n) nếu thỏa :
  - T(n) = O(f(n)), và
  - Nếu ∃ g(n), mà T(n) = O(g(n)) thì f(n) = O(g(n)).

Một số qui tắc về ký hiệu O lớn

- Nếu $f_1(n)=O(g_1(n))$ và $f_2(n)=O(g_2(n))$
  - $f_1(n)+f_2(n)=O(g_1(n)+g_2(n))=max(O(g_1(n),g_2(n))$
  - $f_1(n)*f_2(n)=O(g_1(n)*g_2(n))$
  - $\log_k N=O(N)$ với mọi hằng số k
- Nếu f(n) là một đa thức bậc k,
  - thì $f(n)$ là $O(n^k)$,

Một số qui tắc về ký hiệu O lớn

- $f = O(f)$
- $f = O(g)$ và $g = O(h)$ thì $f = O(h)$
- $f = O(g)$ và $h=O(r)$ thì $fh = O(gr)$
- $f =O(g)$ và $h=O(r)$ thì $f+h = O(g+r)$
- $f =O(g)$ thì $af = O(g)$ với mọi $a>0$.
- $O(c)=O(1)$, c là hằng số

Một số qui tắc về ký hiệu O lớn

- Ví dụ:
  - 2n là O(n)
  - $3n + 5$ là $O(n)$ thay vì $3n + 5$ là $O(3n)$
  - $4n^2 + 5n + 7$ la O(?)

Xác định độ phức tạp tính toán

- T1(n) và T2(n) là thời gian thực hiện của hai giai đoạn chương trình P1 và P2 mà T1(n) = O(f(n)); T2(n) = O(g(n))
- **Qui tắc tổng**:
  - Thời gian thực hiện đoạn P1 rồi P2 tiếp theo sẽ là T1(n) + T2(n) = O(max(f(n),g(n))).
- **Qui tắc nhân**:
  - Thời gian thực hiện P1 và P2 lồng nhau sẽ là : T1(n)T2(n) = O(f(n)*g(n))

Các qui tắc tổng quát

- Các phép gán, đọc, viết, goto là các phép toán sơ cấp:
  - Thời gian thực hiện là: O(1)
- **Lệnh lựa chọn: if-else** có dạng

if (<điều kiện>)

lệnh 1

else

lệnh 2

Các qui tắc tổng quát

- Câu lệnh switch được đánh giá tương tự như lệnh if-else.
- Các lệnh lặp: for, while, do-while
  - Cần đánh giá số tối đa các lần lặp, giả sử đó là L(n)
  - Tiếp theo đánh giá thời gian chạy của mỗi lần lặp là $T_i(n)$, (i=1,2,..., L(n))
  - Mỗi lần lặp, chi phí kiểm tra điều kiện lặp,là $T_0(n)$.
  - Chí phí lệnh lặp là: $$\sum_{i=1}^{L(n)} \left(T_0(n) + T_i(n)\right)$$

Một số ví dụ

- **Ví dụ 1**. Mảng A các số thực, cỡ n, cần tìm xem mảng có chứa số thực x không.

```
(1)  i = 0;
(2)  while (i < n &&  x != A[i])
(3)   i++;
```

Một số ví dụ

Case1:
```
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        k++;
```
$O(n^2)$

Case 2:
```
for (i=0; i<n; i++)
    k++;
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        k++;
```
$O(n^2)$

Case 3:
```
for (int i=0; i<n-1; i++)
    for (int j=0; j<i; j++)
        int k+=1;
```
$O(n^2)$

Một số ví dụ

```
int MaxSubSum1(const int a[], int n) {
    int maxSum=0;

    for (int i=0; i<n; i++)
        for (int j=i; j<n; j++) {
            int thisSum=0;

            for (int k=i; k<=j; k++)
                thisSum+=a[k];

            if (thisSum>maxSum)
                maxSum=thisSum;
        }
    return maxSum;
}
```

$O(n^3)$

Một số ví dụ

```
int MaxSubSum2(const int  a[], int n) {
    int maxSum=0;

    for (int i=0; i<n; i++) {
        thisSum=0;
        for (int j=i; j<n; j++) {
            thisSum+=a[j];

            if (thisSum>maxSum)
                maxSum=thisSum;
        }
    }
    return maxSum;
}
```

$O(n^2)$

Một số ví dụ

```
int MaxSubSum4(const int a[], int n) {
    int maxSum=0, thisSum=0;

    for (int j=0; j<n; j++) {
        thisSum+=a[j];

        if (thisSum>maxSum)
            maxSum=thisSum;
        else if (thisSum<0)
            thisSum=0;
    }
    return maxSum;
}
```

$O(n)$

Một số ví dụ

```
Sum=0
for (j=0;j<N;j++)
  for (k=0;k<N*N;k++)
    Sum++;
```

$O(N^3)$

Một số ví dụ

Ví dụ: sắp xếp dãy

```
void BubbleSort(int a[], int n)
{      int i,j,temp;
(1)      for(i= 0; i<=n-2; i++)
(2)        for(j=n-1; j>=i+1;j--)
(3)          if (a[j] < a[j-1]) {
(4)             temp=a[j-1];
(5)          a[j-1] = a[j];
(6)          a[j]    = temp;
              }
 }
```

Một số ví dụ

- Lệnh (3), (4), (5) và (6) đều tốn O(1)
- Vòng lặp (2) thực hiện (n-i) lần, mỗi lần O(1) do đó vòng lặp (2) tốn O((n-i)*1) = O(n-i).
- Vòng lặp {1},i chạy từ 1 đến n-1, thời gian thực hiện của vòng lặp (1)
- Độ phức tạp của giải thuật là

$$T(n) = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = O(n^2)$$

PHÂN TÍCH CÁC HÀM ĐỆ QUY

- **Ví dụ:** Hàm tính giai thừa

```
int   Fact(int  n)
  {
    if (n <= 1)
      return  1;
    else    return   n * Fact(n-1);
  }
```

- Với n <= 1, ta có T(1) = O(1).
- Với n > 1, ta có T(n) = O(1) + T(n-1)

PHÂN TÍCH CÁC HÀM ĐỆ QUY

- Ta có quan hệ đệ quy sau:
  - $T(1) = O(1)$
  - $T(n) = T(n-1) + O(1)$ với $n > 1$
- Thay các ký hiệu O(1) bởi các hằng số dương a và b tương ứng, ta có
  - $T(1) = a$
  - $T(n) = T(n-1) + b$ với $n > 1$

PHÂN TÍCH CÁC HÀM ĐỆ QUY

- Sử dụng các phép thế T(n-1) = T(n-2) + b, T(n-2) = T(n-3) + b,..., ta có

$$
\begin{aligned}
T(n) &= T(n-1) + b \\
&= T(n-2) + 2b \\
&= T(n-3) + 3b \\
&\;\;\ldots \\
&= T(1) + (n-1)b \\
&= a + (n-1)b
\end{aligned}
$$

Từ đó, ta suy ra T(n) = O(n).

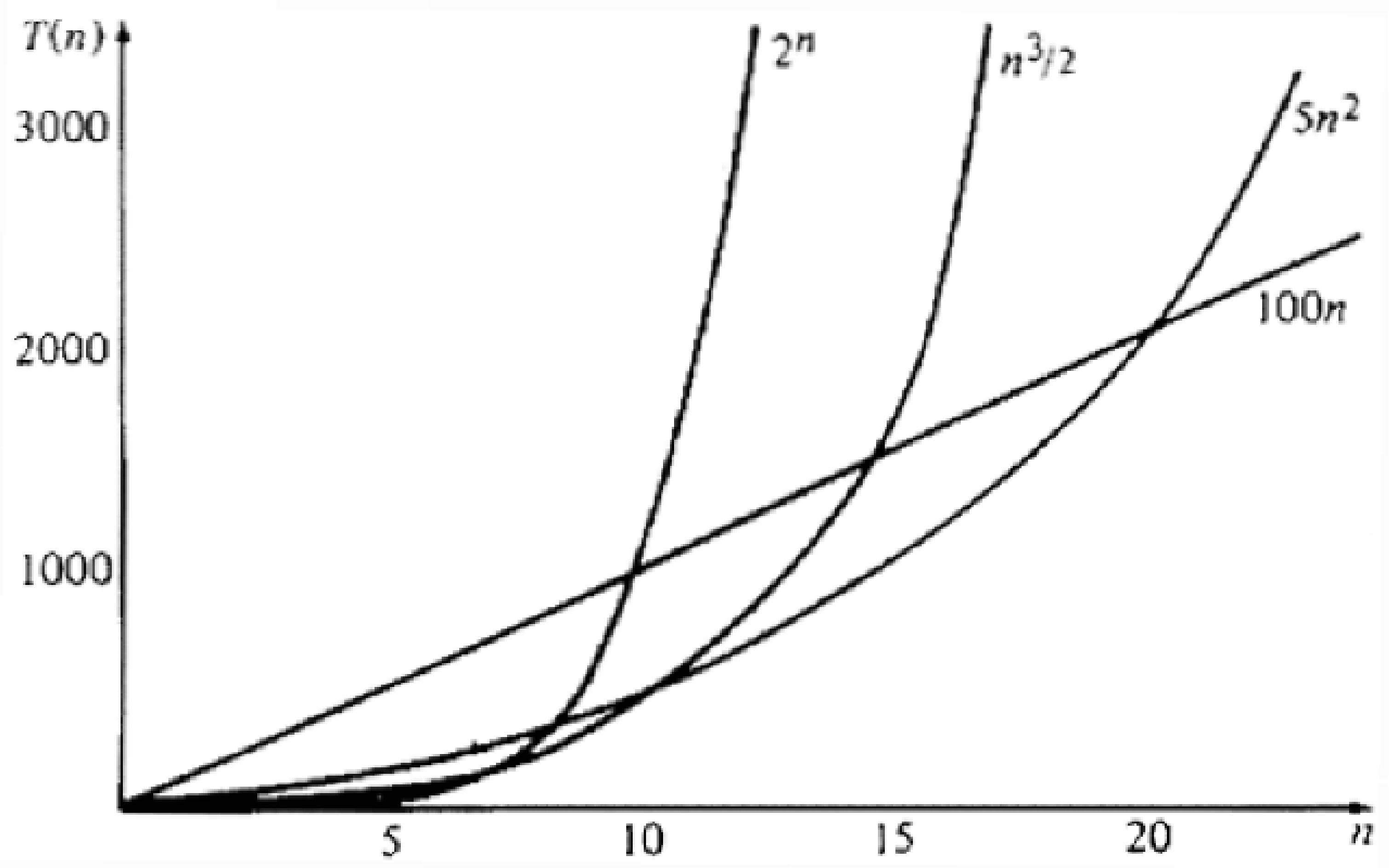
PHÂN TÍCH CÁC HÀM ĐỆ QUY

- Gọi T(n) là thời gian chạy của hàm đệ quy F
- Khi đó, thời gian chạy của các lời gọi hàm ở trong hàm F sẽ là T(m) (với m < n)
- Trước hết, phải đánh giá thời gian chạy của hàm F trên dữ liệu nhỏ nhất n = 1, giả sử T(1) = a (điều kiện dừng)
- Sau đó, đánh giá thời gian chạy của các câu lệnh trong thân của hàm F
- Tìm ra quan hệ đệ quy biểu diễn thời gian chạy của hàm F thông qua lời gọi hàm

Sự phân lớp của giải thuật

- Độ phức tạp
  - O(1) độ phức tạp hằng số
  - O(logn) độ phức tạp logarit
  - O(n) độ phức tạp tuyến tính
  - O(nlogn) độ phức tạp nlogn
  - $O(n^b)$ độ phức tạp đa thức
  - $O(b^n)$ độ phức tạp mũ
  - O(n!) độ phức tạp giai thừa

Sự phân lớp của giải thuật



Đánh giá độ phức tạp trong ba trường hợp

- Độ phức tạp tính toán của giải thuật trong các trường hợp
  - Xấu nhất
  - Tốt nhất
  - Trung bình

Đánh giá độ phức tạp trong ba trường hợp

- Ví dụ 8: Thuật toán tìm kiếm tuần tự

```
int sequenceSearch(int x, int a[], int n){
    for (int i=0;i<n;i++){
        if (x==a[i]) return i;
     }
    return -1;
}
```

Đánh giá độ phức tạp trong ba trường hợp

- Tốt nhất: phần tử đầu tiên là phần tử cần tìm, số lượng phép so sánh là 2 → T(n) ~ O(2) = O(1)
- Xấu nhất: so sánh đến phần tử cuối cùng, số lượng phép so sánh là 2n → T(n) ~ O(n)
- Trung bình: so sánh đến phần tử thứ i, cần 2i phép so sánh, vậy trung bình cần

(2+4+6+…+2n)/n=2(1+2+…+n)/n=n+1

→ T(n) ~ O(n)

Kiến thức Toán học bổ trợ về Tổng các chuỗi

$$S(N) = 1 + 2 + \ldots + N = \sum_{i=1}^{N} i = N(1+N)/2$$

- Tổng các BP: $$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6} \approx \frac{N^3}{3} \text{ for large N}$$

- Logarithms:
  - $x^a = b \Leftrightarrow \log_x b = a$

Kiến thức Toán học bổ trợ về Tổng các chuỗi

$$\sum_{i=1}^{N} i^k \approx \frac{N^{k+1}}{|k+1|} \text{ for large N and k} \neq \text{-1}$$

$$\sum_{i=0}^{N} A^i = \frac{A^{N+1}-1}{A-1}$$

- Đặc biệt khi A = 2
  - $2^0 + 2^1 + 2^2 + \ldots + 2^N = 2^{N+1} - 1$

Q&A