

Tổng Quan Cấu trúc dữ liệu & Giải thuật

Lecturer: Duc-Hieu Tran

Title: MSc. Computer Science

Nội dung

Vai trò của việc tổ chức dữ liệu

Khái niệm về Cấu trúc dữ liệu

Khái niệm về thuật giải

Mối quan hệ giữa Thuật giải và Cấu trúc dữ liệu

Thiết kế Giải thuật

Đánh giá độ phức tạp Thuật giải

Nhắc lại

❖ Kỹ thuật lập trình (Programming Technique)

▪ Giải thuật

- Khái niệm
- Tính chất

▪ PTTK/lập trình hướng đối tượng

- Các thành phần
- Khái niệm
- Tính chất

Nhắc lại

❖ Kỹ thuật lập trình (Programming Technique)

▪ Giải thuật

- Là một **tập các chỉ dẫn** (instruction) để **hướng dẫn** máy tính đi **giải quyết** một vấn đề nào đó hoặc **thực hiện** một công việc cụ thể nào đó
- Tính chất: **đúng đắn, tổng quát, xác định, hiệu quả, hữu hạn**
- Ví dụ: tìm số lớn nhất giữa hai số, ra chợ mua bó rau

Nhắc lại

❖ Kỹ thuật lập trình (Programming Technique)

▪ PTTK/lập trình hướng đối tượng

- **Lớp (class)**: một **khái niệm trừu tượng** thể hiện sự **đại diện** cho một tập hợp các đối tượng có cùng chung **đặc điểm** và **hành vi**
 - Ví dụ: lớp động vật có vú, lớp động vật biết bay, lớp phương tiện, lớp nhân viên, lớp sinh viên
- **Đối tượng (object)**: một thực thể **cụ thể** trong thế giới **thực**
 - Ví dụ: con mèo, con chó, con người, SV Nguyễn Văn A, con mèo tam thể, con chó bec-giê

Nhắc lại

❖ Kỹ thuật lập trình (Programming Technique)

▪ PTTK/lập trình hướng đối tượng

○ Khái niệm

- **Mô hình hóa** vấn đề/bài toán trong thế giới thực
- Bài toán được **biểu diễn** dưới dạng một tập hợp các **đối tượng**
- Giải quyết bài toán là đi **giải quyết** các **vấn đề của đối tượng** đó

Nhắc lại

❖ Kỹ thuật lập trình (Programming Technique)

▪ PTTK/lập trình hướng đối tượng

○ Lớp: thành phần dữ liệu + thành phần xử lý

➤ **Thành phần dữ liệu:** mô tả/lưu trữ các đặc điểm của đối tượng → thuộc tính (property), tính chất (attribute)

➤ **Thành phần xử lý:** thể hiện các hành vi, xây dựng cách thức xử lý các vấn đề của đối tượng đó → phương thức (method)

Nhắc lại

❖ Kỹ thuật lập trình (Programming Technique)

▪ PTTK/lập trình hướng đối tượng

○ Tính chất đặc trưng

➤ Tính đóng gói (encapsulation)

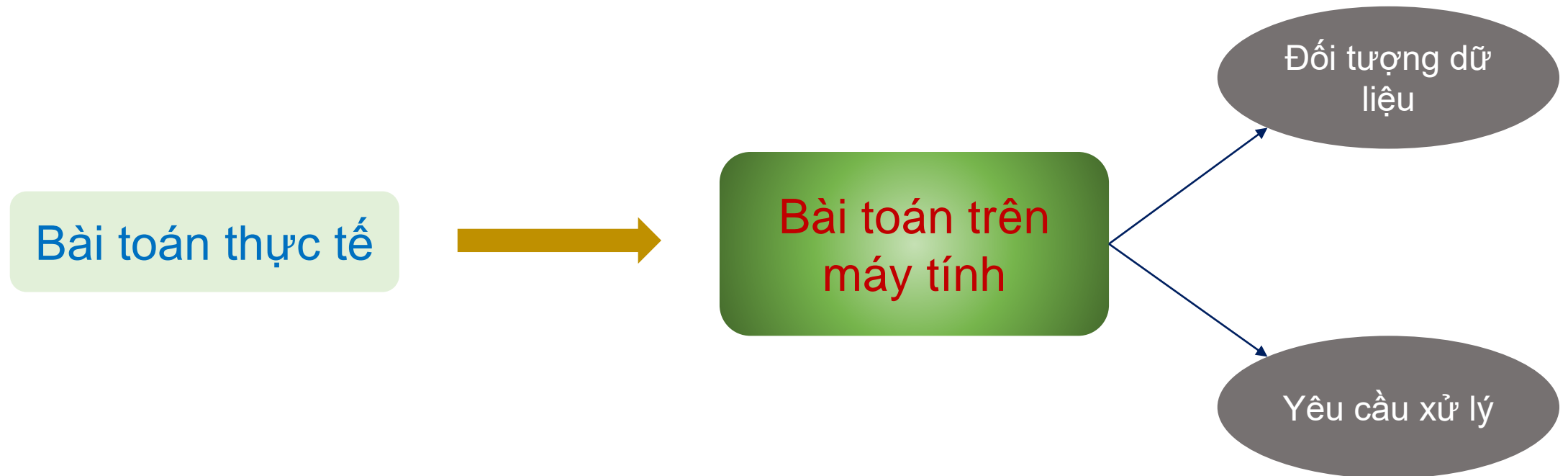
➤ Tính kế thừa (inheritance)

➤ Tính đa hình (polymorphism)

➤ Tính trừu tượng (abstraction)

Vai trò Tổ chức dữ liệu

Giải quyết bài toán bằng máy tính?



Giải quyết bài toán bằng máy tính?

❖ Ví dụ: Xây dựng phần mềm nhận dạng biển số xe



Giải quyết bài toán bằng máy tính?

❖ Ví dụ: Xây dựng website bán hàng trực tuyến

The screenshot displays the Sapo admin dashboard. The left sidebar contains navigation links: Tổng quan, Đơn hàng, Vận chuyển, Khách hàng, Sản phẩm, Số quỹ, Báo cáo, KÊNH BÁN HÀNG (Bán tại quầy, Website, Kênh Facebook, Sàn TMĐT), Ứng dụng, and Cấu hình. The main content area is titled 'Xin chào Trần Đức Hiếu đến với Sapo!' and includes a welcome message. Below this, there are four steps to get started:

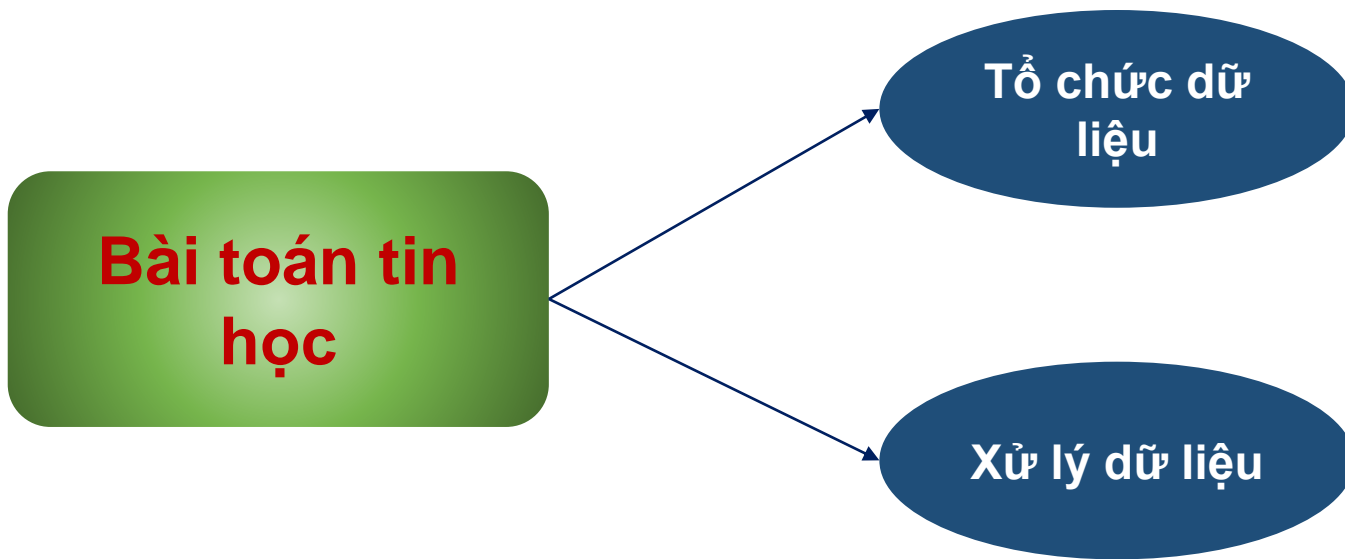
- 1. Cấu hình cửa hàng**: Cài đặt các thông tin cho việc bán hàng của bạn. Button: Cấu hình cửa hàng.
- 2. Thêm sản phẩm**: Tạo mới và quản lý tồn kho theo từng phiên bản sản phẩm. Button: Thêm sản phẩm mới.
- 3. Kết nối đa kênh**: Kết nối với kênh bán hàng hiện tại của bạn để quản lý tập trung tại Sapo.
- 4. Sẵn sàng bán hàng**: Tất cả đã sẵn sàng. Tạo đơn và quản lý đơn hàng của bạn. Button: Tạo đơn hàng mới.

At the bottom, there is a banner for the Sapo mobile app with the text 'Tải ứng dụng miễn phí Sapo cho smartphone' and 'Giúp bạn quản lý bán hàng, gọi ship giá rẻ'. It includes QR codes and links to the App Store and Google Play. The footer shows a countdown timer 'Bạn còn 15 ngày dùng thử miễn phí', a button 'Kích hoạt gói dịch vụ ngay', a support icon, and contact information: 'Liên hệ tư vấn CỘNG HỖ TRỢ SAPO' with a phone number '1800 6750' and a 'Chat Messenger' button.

Giải quyết bài toán bằng máy tính?

❖ 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 quyết bài toán bằng máy tính?

❖ Hai yếu tố tạo nên một chương trình máy tính (Niklaus Wirth, 1976)

Data Structures + **Algorithms** = Programs

Khái niệm Cấu trúc dữ liệu

Dữ liệu

❖ Khái niệm

- **Dữ kiện** có được do quan sát, nhận thức, thu thập
- Được **biểu diễn** dưới dạng những con số, hình ảnh, kí hiệu, kí tự, ghi chép,...

❖ Đặc tính

- Phản ánh giá trị của dữ kiện trong thực tế
- Không bao hàm một ý nghĩa cụ thể

Dữ liệu

❖ Ví dụ

- 8, 15, 11, 13, 9
- Saigon Heat, Singapore Slings

Dữ liệu

❖ Ví dụ



Thông tin

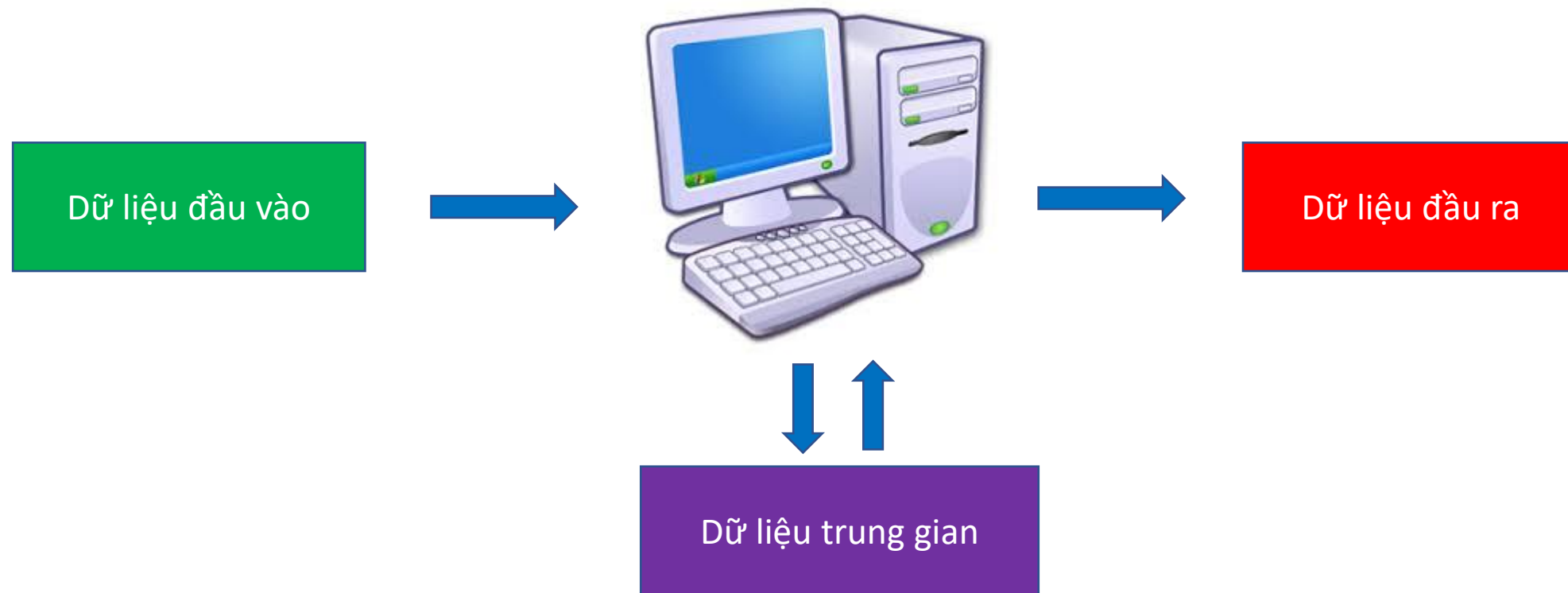
❖ Khái niệm

- Một sự kết hợp của tập các **dữ liệu**
- Được **tổ chức** và **xử lý** để biểu đạt một **ý nghĩa** nào đó

❖ Ví dụ

- 8, 15, 11, 13, 9, 12 và 10 là **điểm số** của một trận đấu bóng rổ giữa 2 đội **Saigon Heat** và **Singapore Slingers** trong khuôn khổ vòng loại **VBA**

Mô hình xử lý dữ liệu



Cấu trúc dữ liệu là gì?



Các cuốn sách chưa được tổ chức, sắp xếp



Các cuốn sách đã được tổ chức, sắp xếp

Cấu trúc dữ liệu

❖ Khái niệm

- Là phương thức để **tổ chức**, **lưu trữ** và **truy xuất** dữ liệu một cách có **hiệu quả**

❖ Xác định cấu trúc dữ liệu

- Tìm cách **biểu diễn** các **dữ liệu** của bài toán thực tế vào máy tính

Cấu trúc dữ liệu

❖ **Ví dụ:** Tìm kiểu dữ liệu để lưu trữ cho các dữ liệu sau

- 452
- 63.2
- Có 3 loại Attentions: Self-attention trong encoder, attention giữa encoder-decoder, (masked) self-attention trong decoder (training)
- 34, 656, 2, 8, 19, 3, 2, 7, 8
- H
- Sinh Viên (“Nguyễn Văn A”, “21DTHXA”, “SV3482930283”, “KTLT”, 8.5)

Cấu trúc dữ liệu

❖ **Ví dụ:** Tìm kiểu dữ liệu để biểu diễn cho bài toán sau

Cho một chuỗi A và một chuỗi B. Chuỗi B có độ dài bé hơn hoặc bằng chuỗi A. Hãy viết chương trình để xác định xem chuỗi B có xuất hiện trong chuỗi A hay không ? Nếu có thì hãy xuất ra vị trí xuất hiện đầu tiên của chuỗi B trong chuỗi A. Nếu không thì hãy xuất ra giá trị -1

Cấu trúc dữ liệu

❖ Tiêu chuẩn của một cấu trúc dữ liệu

- Biểu diễn đầy đủ thông tin
- Phù hợp với các thao tác xử lý
- Phù hợp với điều kiện cho phép của ngôn ngữ lập trình
- Tiết kiệm tài nguyên hệ thống

Vai trò cấu trúc dữ liệu

- ❖ Cấu trúc dữ liệu (CTDL) đóng vai trò quan trọng trong việc kết hợp thuật giải để đưa ra cách giải quyết bài toán
- ❖ CTDL phải biểu diễn được đầu vào (Input) và đầu ra (Output) của bài toán
- ❖ CTDL hỗ trợ cho các thuật toán thao tác trên các đối tượng được hiệu quả hơn
- ❖ Sự lựa chọn thuật toán để giải quyết vấn đề cũng phụ thuộc rất nhiều vào CTDL được sử dụng

Ví dụ

- ❖ Một chương trình quản lý điểm thi của sinh viên cần lưu trữ các điểm số 4 môn của 3 sinh viên tương ứng như sau:

Sinh viên	Môn 1	Môn 2	Môn 3	Môn 4
SV1	7	9	5	2
SV2	5	0	9	4
SV3	6	3	7	4

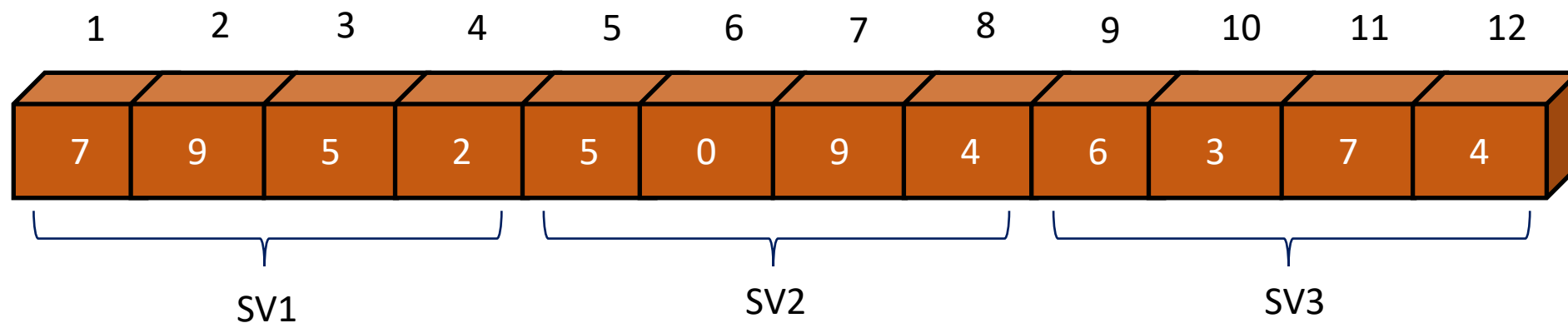
Bài toán đặt ra: biểu diễn các dữ liệu trên vào máy tính và xuất điểm số các môn học của từng sinh viên

Minh họa về cấu trúc dữ liệu

❑ Phương án 1: sử dụng mảng 1 chiều

Có tất cả $3(\text{SV}) * 4(\text{Môn}) = 12$ điểm số cần lưu trữ, do đó ta có mảng như sau:

```
int[] score = {7, 9, 5, 2, 5, 0, 9, 4, 6, 3, 7, 4};
```



⇒ Muốn xác định được điểm số một môn học của sinh viên thì ta phải áp dụng công thức: Điểm số $(\text{SV}_i, \text{Môn}_j) = \text{score}[(i - 1) * \text{số môn} + j]$

Ví dụ: Điểm số $(\text{SV2}, \text{Môn 3}) = \text{score}[(2-1) * 4 + 3] = \text{score}[7] = 9$

Minh họa về cấu trúc dữ liệu

❑ **Phương án 2:** sử dụng mảng 2 chiều

Có tất cả 3(SV) * 4(Môn) \Rightarrow sử dụng mảng 3 * 4 như sau:

```
int[][] score = {{7, 9, 5, 2}, {5, 0, 9, 4}, {6, 3, 7, 4}};
```

	Cột 1	Cột 2	Cột 3	Cột 4
Dòng 1	7	9	5	2
Dòng 2	5	0	9	4
Dòng 3	6	3	7	4

\Rightarrow Muốn xác định được điểm số một môn học của sinh viên thì ta chỉ cần truy xuất đúng số dòng và số cột của phần tử trong bản:

Điểm số (SV_i , $Môn_j$) = score[i, j];

Khái niệm Thuật Giải

Thuật Giải

❖ **Khái niệm:** Thuật giải là một tập **hữu hạn** có thứ tự các bước tác động lên **dữ liệu** để sau một số lần thực hiện sẽ cho ra **kết quả**



Đặc trưng

- ❖ Có dữ liệu **Đầu vào** (Input): nhận được dữ liệu đầu vào
- ❖ Có dữ liệu kết quả **Đầu ra** (Output): sinh được dữ liệu đầu ra
- ❖ Có tính **Chính xác / Đúng đắn** (Precision): các bước mô tả để thực hiện là đúng
- ❖ Có tính **Hữu hạn** (Finiteness): sau một số lần thực hiện, giải thuật phải kết thúc
- ❖ Có tính **Duy nhất** (Uniqueness): cùng một dữ liệu đầu phải cùng một kết quả đầu ra
- ❖ Có tính **Hiệu quả** (Efficiency): giải quyết được công việc trong thời gian cho phép
- ❖ Có tính **Tổng quát** (Generality): có thể thực hiện được với mọi dữ liệu phù hợp

Cách biểu diễn

- ❖ Ngôn ngữ tự nhiên (Natural Language)
- ❖ Lưu đồ (Flow chart)
- ❖ Mã giả (Pseudo code)
- ❖ Ngôn ngữ lập trình (Programming Language)

Mối quan hệ CTDL & GT

- ❖ Đối tượng xử lý của giải thuật chính là **dữ liệu**
- ❖ Với mỗi cấu trúc dữ liệu, sẽ có những giải thuật tương ứng
- ❖ Cấu trúc dữ liệu thay đổi → giải thuật thay đổi theo



Sự cần thiết của Thuật giải?

❖ Tại sao phải sử dụng máy tính để xử lý dữ liệu?

- Nhanh hơn, chính xác hơn
- Giải quyết được nhiều bài toán đòi hỏi khối lượng tính toán cực lớn, hoặc những bài toán phức tạp với khối lượng dữ liệu lớn

➤ Tuy nhiên, bài toán lớn đòi hỏi phải có máy tính hiệu quả để thực hiện

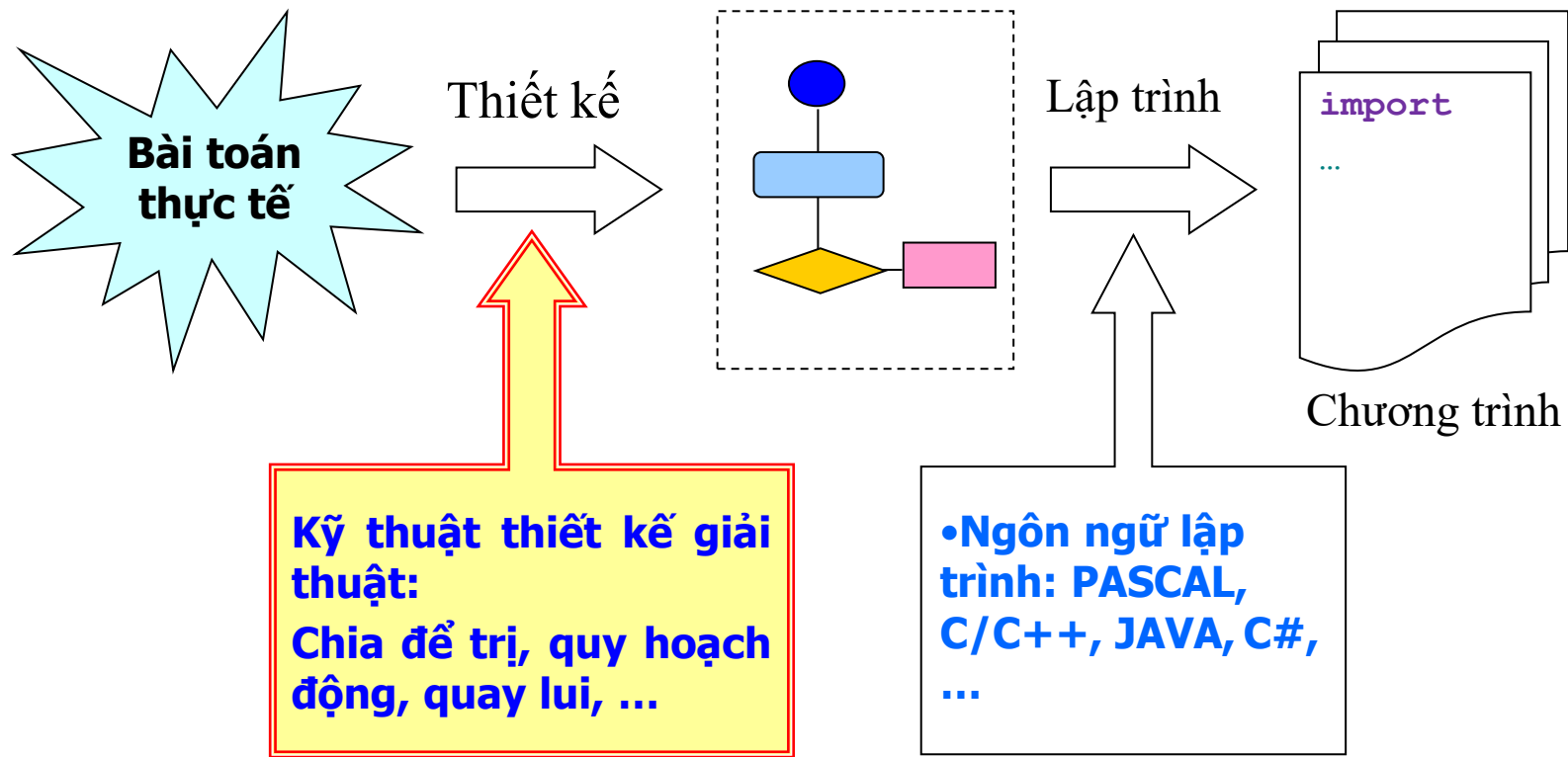
❖ Phương pháp?

- Sử dụng máy tính có cấu hình mạnh → chi phí cao
- Xây dựng thuật toán xử lý hiệu quả → thông minh, chi phí thấp

Thiết kế Giải Thuật

Thiết kế giải thuật

❖ Từ bài toán đến chương trình



Thiết kế giải thuật

❖ 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)
- ...

Nguyên lý thiết kế Top-down

❖ Đi từ trên xuống (Top-Down Design)

- Dựa trên ý tưởng của chiến lược **chia để trị**
- Chia bài toán lớn thành các bài toán **nhỏ** (bài toán con)
 - Nếu bài toán nhỏ **đơn giản** \Rightarrow giải trực tiếp
 - Nếu **chưa giải được** bài toán nhỏ \Rightarrow tiếp tục chia

Gộp kết quả của các bài toán con để xây dựng kết quả của bài toán ban đầu

❖ Có thể áp dụng với các chiến lược: chia để trị, quy hoạch động, quay lui

Nguyên lý thiết kế Top-down

❖ Lược đồ chung

DivideAndConquer(A , x) // tìm nghiệm x của bài toán A

{

 if (A đủ nhỏ)

Solve(A); // Giải quyết

 else {

 Chia bài toán A thành các bài toán con A_1, A_2, \dots, A_m ;

 for ($i = 1$; $i \leq m$; $i++$)

DivideAndConquer(A_i , x_i); // tìm nghiệm x_i của bài toán con A_i

 Kết hợp các nghiệm x_i của các bài toán con A_i ($i = 1, \dots, m$) để nhận được nghiệm x của bài toán A ;

 }

}

Nguyên lý thiết kế Top-down

❖ **Ví dụ:** Tìm phần tử lớn nhất trong một dãy có n số

- Gọi A là dãy số dữ liệu đầu vào (A xác định)
- Gọi x là phần tử có giá trị lớn nhất trong dãy A cần tìm

Nguyên lý thiết kế Top-down

❖ Ví dụ: Tìm phần tử lớn nhất trong một dãy có n số

FindMaxValue(A, x) // tìm x trong dãy số A

```
{  
    if ( $A.length == 1$ ) //  $A$  chỉ có 1 phần tử  
        return  $A[0]$ ; //  $x$  là phần tử duy nhất trong  $A$   
    else {  
        Chia dãy số  $A$  thành 2 dãy  $A_1, A_2$   
         $x_1 = \text{FindMaxValue}(A_1, x_1)$ ; // tìm  $x_1$  của dãy  $A_1$   
         $x_2 = \text{FindMaxValue}(A_2, x_2)$ ; // tìm  $x_2$  của dãy  $A_2$   
        if ( $x_1 > x_2$ ) return  $x_1$ ; // kết hợp nghiệm  $x_1, x_2$  để suy ra  $x$   
        else return  $x_2$ ;  
    }  
}
```

Nguyên lý thiết kế Top-down

❖ Ví dụ: Tìm phần tử lớn nhất trong một dãy có n số

- Chia dãy ban đầu thành 2 dãy A_1, A_2 có $n/2$ phần tử. Tìm phần tử lớn nhất trong A_1 và so sánh với phần tử lớn nhất trong $A_2 \Rightarrow$ phần tử lớn nhất của dãy số
- Nếu chưa tìm được phần tử lớn nhất trong $A_1 \Rightarrow$ tiếp tục chia A_1 thành 2 dãy con nhỏ hơn, ...
- Nếu chưa tìm được phần tử lớn nhất trong $A_2 \Rightarrow$ tiếp tục chia A_2 thành 2 dãy con nhỏ hơn...

Nguyên lý thiết kế Bottom-up

❖ Đi từ dưới lên (Bottom-up Design)

- Dựa trên khái niệm **qui nạp toán học**
- Cách thực hiện:
 - Đi **giải quyết** trực tiếp **bài toán nhỏ** với số lượng dữ liệu là **ít nhất**
 - **Thêm** dữ liệu vào bài toán nhỏ **để mở rộng** ra một chút và giải quyết
 - **Tiếp tục mở rộng** cho đến khi bài toán nhỏ có kích thước **bằng** với bài toán lớn
 - Kết quả sau cùng chính là kết quả của **bài toán lớn**
- ❖ Có thể áp dụng với các chiến lược: quy hoạch động, tham lam, tìm kiếm từng bước

Nguyên lý thiết kế Bottom-up

❖ Ví dụ: Tìm giá trị của phần tử lớn nhất trong một dãy có n số

- Gọi **max** là giá trị phần tử lớn nhất cần tìm
- Ban đầu gán **max** = giá trị phần tử **[0]**
- Đưa tiếp phần tử **[1]** vào so sánh với max
 - Nếu giá trị **[1]** > **max** \Rightarrow cập nhật **max** = giá trị **[1]**
- Cứ thế làm với các phần tử **[2], [3], ..., [n]**

Sau khi xét hết \Rightarrow **max** chính là giá trị của phần tử **lớn nhất** trong dãy

Bài tập

❖ Bài tập: Dãy số Fibonacci

- Dãy Fibo bắt đầu với 2 phần tử $F[0] = 1, F[1] = 1$
- Các phần tử tiếp theo được tính bằng công thức $F[i] = F[i-1] + F[i-2]$ ($i \geq 2$)
- Cho một số n ($n \geq 2$), hãy xác định giá trị của phần tử $F[n]$
- ❖ Ví dụ: $n = 5 \Rightarrow F[n] = 8$
- ❖ Ví dụ: $n = 7 \Rightarrow F[n] = 21$
- **Yêu cầu:** Xây dựng thuật toán giải bài toán trên theo cả 2 nguyên lý thiết kế

Đánh giá Độ Phức Tạp Thuật Giải

Lựa chọn Thuật giải

❖ Giải quyết bài toán

- Đứng trước việc phải lựa chọn giải thuật nào?
- Dựa trên cơ sở nào để lựa chọn?

⇒ Lựa chọn thuật giải có tính hiệu quả cao nhất

❖ Tính hiệu quả của thuật giải được đo đặc thông qua các tiêu chí

- Có chi phí sử dụng tài nguyên thấp: ít tốn bộ nhớ, ít thời gian sử dụng CPU,...
- Có thời gian thực hiện giải thuật nhanh
- Có độ phức tạp giải thuật thấp

Phương pháp Thực nghiệm

- ❖ Thời gian thực hiện giải thuật có thể đo được bằng cách **thực nghiệm**
- ❖ **Ví dụ:** Xây dựng một chuỗi có 10000 kí tự '*'

```
public class StringBuilder {  
    public static void main(String[] args) {  
        long startTime = System.currentTimeMillis(); // Ghi nhận thời điểm bắt đầu  
  
        // Thực hiện giải thuật  
        String result = "";  
        for (int i=0; i < 10000; i++)  
            result = result + "*";  
  
        long endTime = System.currentTimeMillis(); // Ghi nhận thời điểm kết thúc  
        long elapsed = endTime - startTime; // Tính toán thời gian thực hiện giải thuật  
    }  
}
```

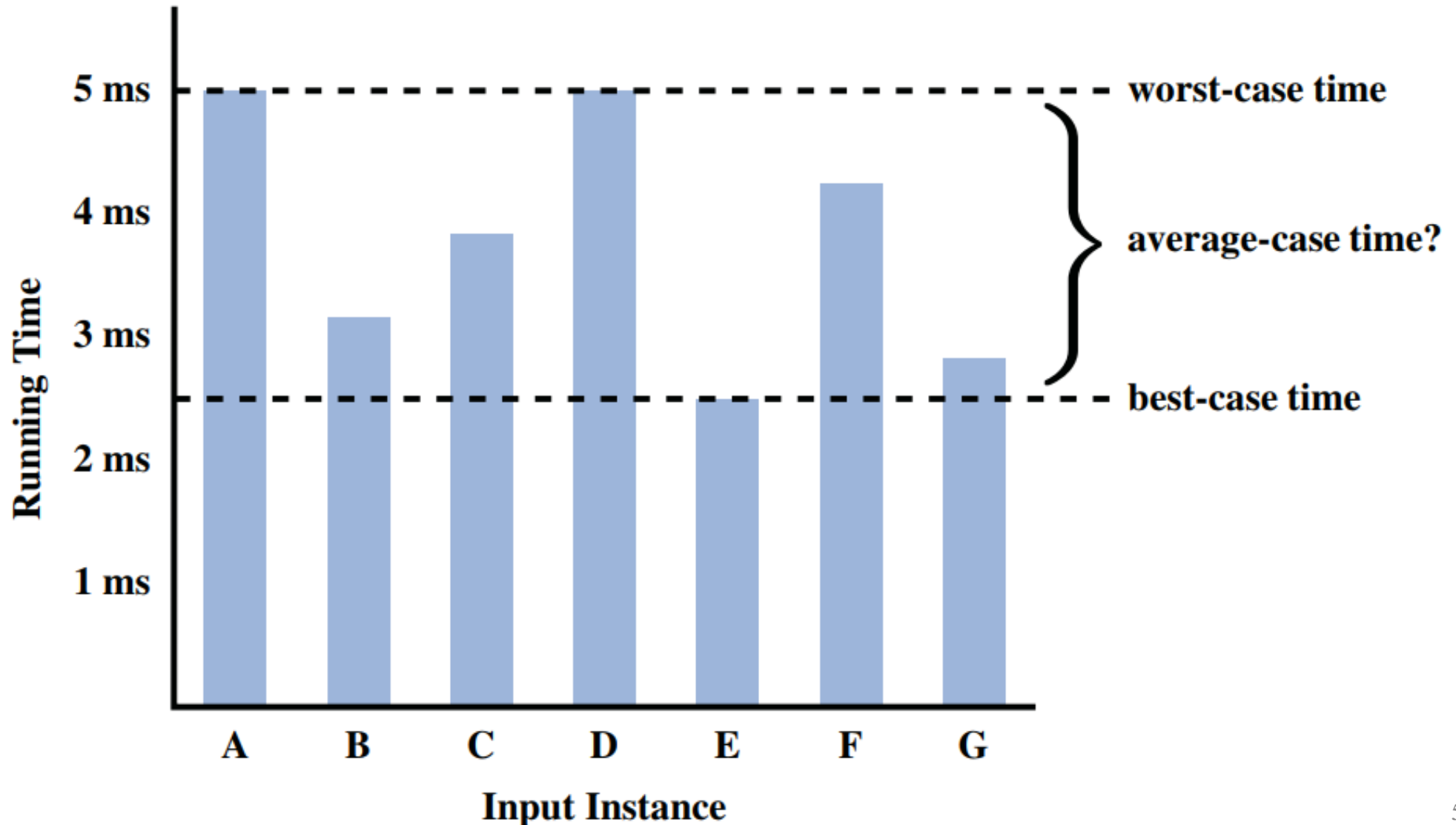
Phương pháp Thực nghiệm

❖ Tuy nhiên, phương pháp **thực nghiệm** này có nhiều **khuyết điểm**:

- Chịu sự **hạn chế** của **ngôn ngữ lập trình** sử dụng
- **Ảnh hưởng** bởi **trình độ** của người cài đặt
- **Không chọn được** bộ **dữ liệu thử** đặc trưng cho tất cả các trường hợp
- **Phụ thuộc** vào **tốc độ xử lý** của máy tính sử dụng

⇒ Kết quả tính toán sẽ **không đồng nhất** nhau khi thay đổi một trong các yếu tố trên

Phương pháp Thực nghiệm



Phương pháp Toán học

- ❖ Là phương pháp đánh giá mang tính tổng quát hơn
 - Ít phụ thuộc môi trường
 - Ít phụ thuộc vào phần cứng
- ❖ Đánh giá thuật toán theo hướng xấp xỉ tiệm cận qua các hàm $O()$, $o()$, $\Omega()$ và $\Theta()$

Phương pháp Toán học

❖ **Nguyên lý:** Đếm **số phép toán cơ bản** để thực hiện giải thuật

❖ **Phép toán cơ bản** là phép toán thực hiện được trong **1 đơn vị thời gian**:

- Gán giá trị cho 1 biến. **Ví dụ:** `int a = 1;`
- Truy xuất thành phần của 1 đối tượng. **Ví dụ:** `Object.x`, `Object.getValue();`
- Thực hiện một phép toán số học. **Ví dụ:** $a + b$, $a * b$, $a - b$, $a / b, \dots$
- So sánh hai số. **Ví dụ:** $a > b$, $a \leq b, \dots$
- Truy cập đến một phần tử của mảng. **Ví dụ:** `a[1]`, `a[6]`, `a[i], \dots`
- Thực hiện một lời gọi hàm. **Ví dụ:** `Print(); GoTo(); GetHeight();`
- Trả về giá trị của hàm. **Ví dụ:** `return 35;` `return result;`

Tính độ phức tạp giải thuật

- ❖ Gọi n là kích thước đầu vào của dữ liệu
- ❖ Thời gian thực hiện của giải thuật là một hàm $f(n)$
- ❖ **Ví dụ:** thời gian chạy của 2 giải thuật

```
1 i = 1
2 loop (i <= 1000)
3     application code
4     i = i + 1
```

$$f(n) = 1000$$

```
1 i = 1
2 loop (i <= 1000)
3     application code
4     i = i + 2
```

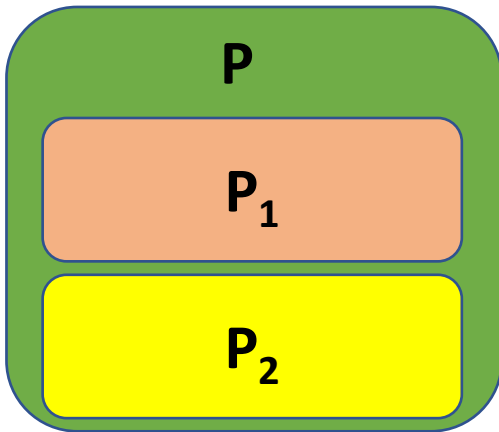
$$f(n) = 500$$

Tính độ phức tạp giải thuật

❖ Hàm $f(n)$ tuân theo 2 nguyên tắc:

❖ Nguyên tắc cộng

Nếu chương trình P gồm 2 đoạn chương trình con $P_1(f_1(n))$ và $P_2(f_2(n))$ chạy nối tiếp nhau

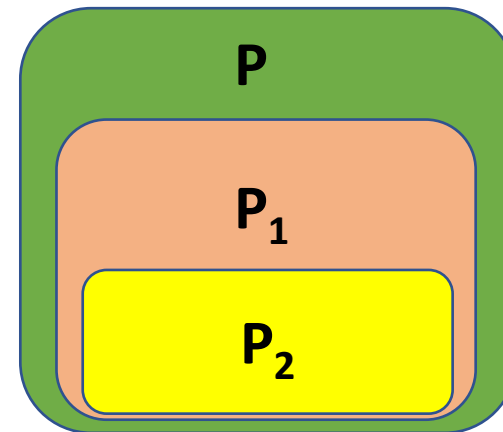


Thì độ phức tạp của P:

$$f(n) = \max(f_1(n), f_2(n))$$

❖ Nguyên tắc nhân

Nếu chương trình P gồm 2 đoạn chương trình con $P_1(f_1(n))$ và $P_2(f_2(n))$ chạy lồng nhau

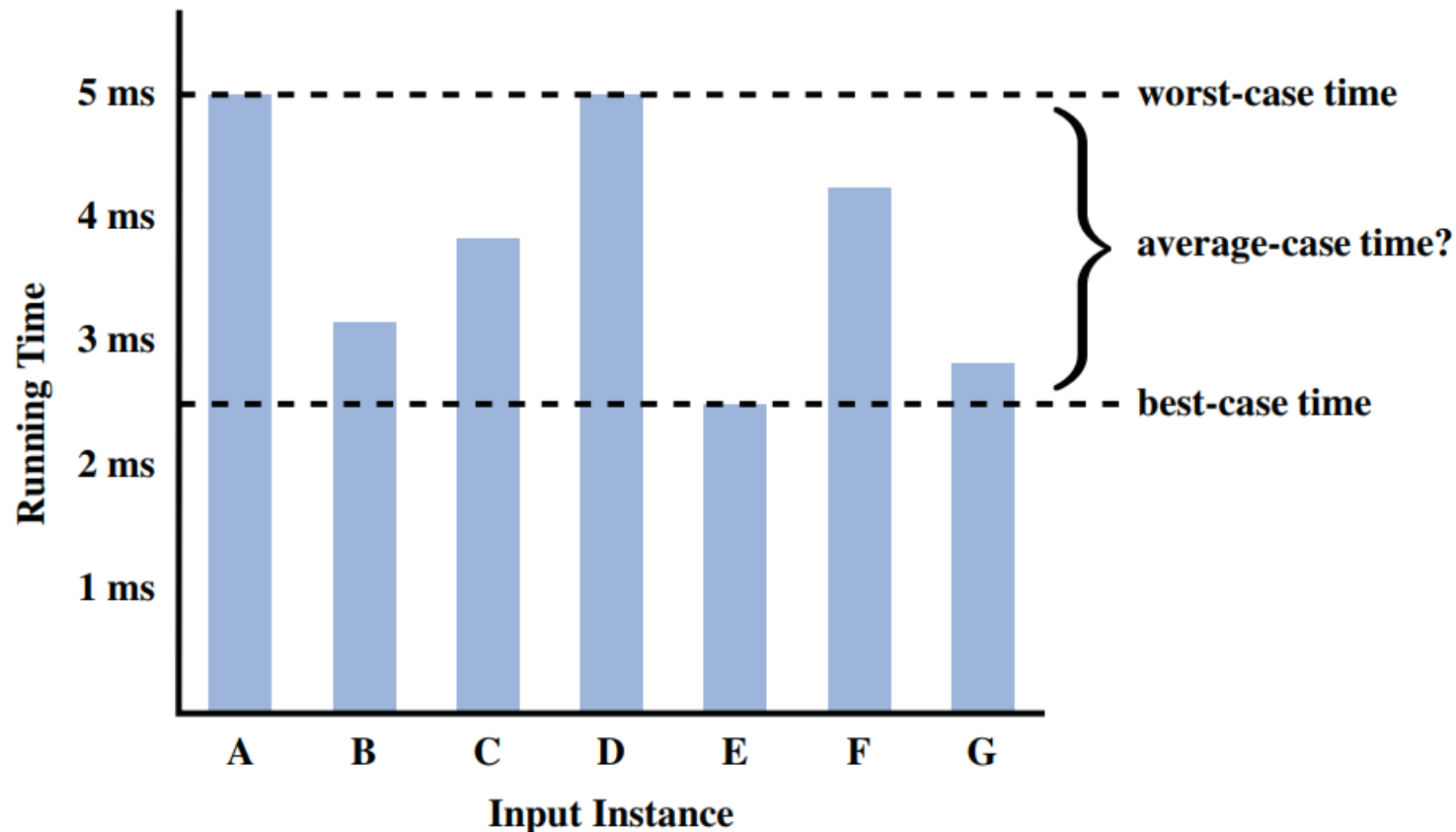


Thì độ phức tạp của P:

$$f(n) = f_1(n) * f_2(n)$$

Tính độ phức tạp giải thuật

□ $f(n)$ là một hàm phụ thuộc vào **dữ liệu đầu vào** nên khi dữ liệu thay đổi thì $f(n)$ cũng thay đổi \Rightarrow chỉ quan tâm đến độ phức tạp của $f(n)$ trong trường hợp xấu nhất



Tính độ phức tạp giải thuật

❖ Khi kích thước đầu vào của dữ liệu tăng \Rightarrow thời gian tính toán cũng tăng
 \Rightarrow chúng ta cần quan tâm đến **tốc độ tăng (rate of growth)** của hàm $f(n)$

❖ Ví dụ

- Giải thuật A, độ phức tạp thời gian $f_A(n)$
- Giải thuật B, độ phức tạp thời gian $f_B(n)$
- Ban đầu $f_A(n) = f_B(n)$
- Khi n lớn, $f_A(n) \gg f_B(n)$

\Rightarrow Giải thuật A chậm hơn giải thuật B

Tính độ phức tạp giải thuật

❖ Tóm lại

➤ Để đánh giá độ phức tạp của giải thuật

- ❑ Đánh giá $f(n)$ trong trường hợp xấu nhất
- ❑ Đánh giá tốc độ tăng của hàm $f(n)$ khi dữ liệu có kích thước lớn

Tính độ phức tạp giải thuật

- ❖ Để xác định độ tăng của hàm $f(n)$ ta sử dụng khái niệm toán học Big-O để chỉ cận trên của $f(n)$
- ❖ **Định nghĩa:** gọi $f(n)$ và $g(n)$ là 2 hàm số không âm của n . Ta có độ tăng của $f(n)$ là $O(g(n))$ nếu tồn tại hằng số c và n_0 sao cho $f(n) \leq c.g(n) \forall n > n_0$
- ❖ **Ý nghĩa:** $g(n)$ được gọi là hàm tiệm cận trên của $f(n)$

Tính độ phức tạp giải thuật

❖ **Ví dụ:** Giả sử $f(n) = 5n^3 + 2n^2 + 13n + 6$, ta có:

$$f(n) = 5n^3 + 2n^2 + 13n + 6 \leq 5n^3 + 2n^3 + 13n^3 + 6n^3 = 26n^3$$

Đặt $g(n) = n^3$ thì ta có $f(n) \leq 26.g(n) \forall n > 1$ (lúc này $c = 26, n_0 = 1$)

Nói cách khác độ phức tạp của hàm $f(n)$ là $O(n^3) \Leftrightarrow f(n)$ là Big-O của n^3

Ý nghĩa: tốc độ tăng của hàm $f(n)$ không lớn hơn hàm $g(n) = n^3$

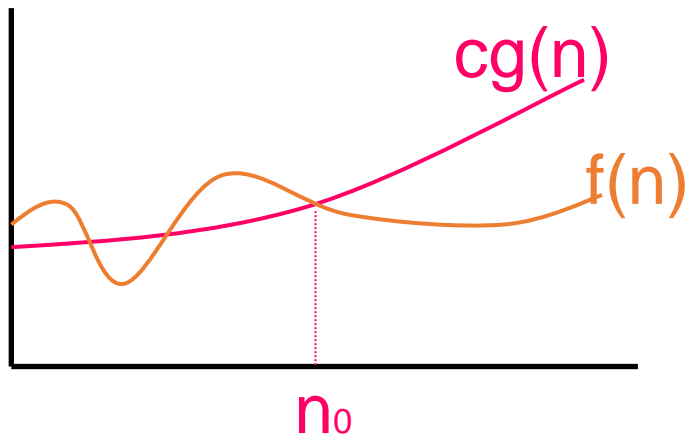
Tính độ phức tạp giải thuật

- ❖ Trong trường hợp ta tìm thấy $f(n) \geq c.g(n)$, $\forall n \geq n_0$ thì ta nói độ phức tạp thấp nhất của $f(n)$ là $\Omega(g(n))$ hay $g(n)$ là $O(f(n))$ (Khái niệm về Ω - Big-Omega) $\Leftrightarrow g(n)$ là tiệm cận dưới của $f(n)$
- ❖ Còn nếu hai hàm $f(n)$ và $g(n)$ tăng trưởng ở một mức độ như nhau, tức là: $c'g(n) \leq f(n) \leq c''g(n)$ thì ta nói $f(n)$ là $\Theta(g(n))$ ($f(n)$ là Big-Theta của $g(n)$) $\Leftrightarrow g(n)$ là tiệm cận của $f(n)$

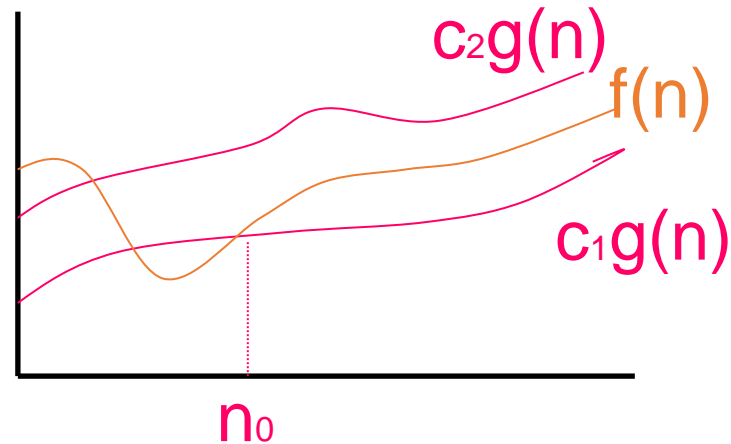
Tính độ phức tạp giải thuật

❖ Biểu đồ thể hiện mối tương quan của $f(n)$ với $O(g(n))$, $\Omega(g(n))$ và $\Theta(g(n))$

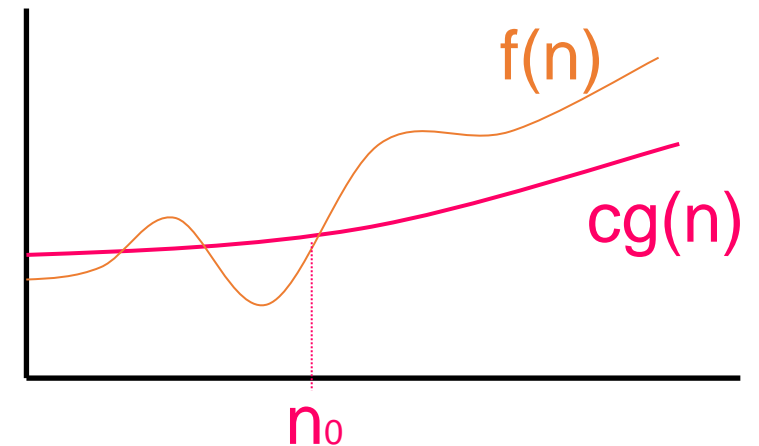
$$f(n) = O(g(n))$$



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



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



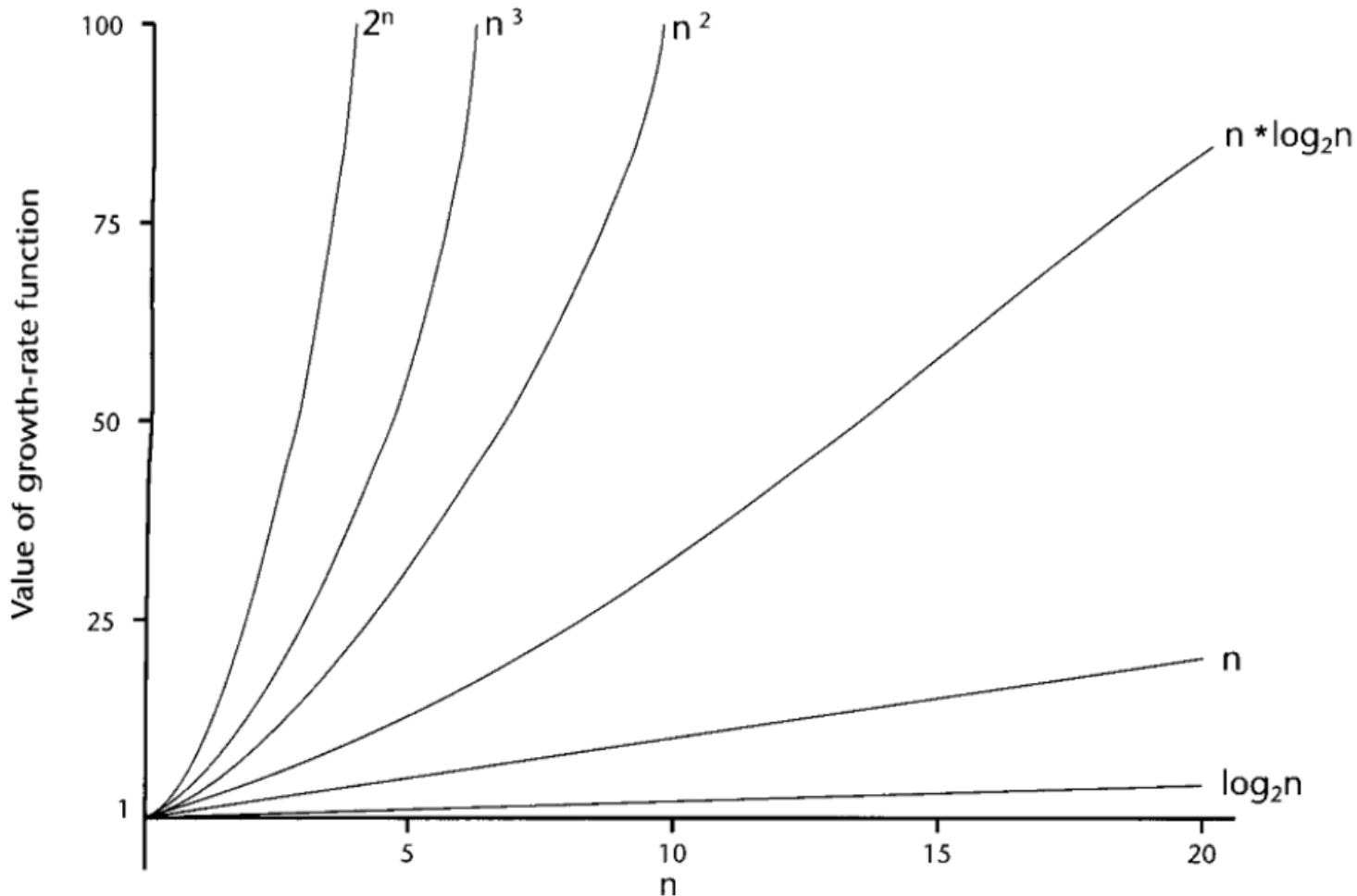
Tính độ phức tạp giải thuật

- ❖ Bảng thể hiện độ tăng của các hàm số ứng với dữ liệu đầu vào có kích thước là n

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84×10^{19}
128	7	128	896	16,384	2,097,152	3.40×10^{38}
256	8	256	2,048	65,536	16,777,216	1.15×10^{77}
512	9	512	4,608	262,144	134,217,728	1.34×10^{154}

Tính độ phức tạp giải thuật

❖ Biểu đồ thể hiện độ tăng của các hàm



17/10/2022

Độ phức tạp tăng dần

- ❖ $n!$: $O(n!)$
- ❖ 2^n : $O(2^n)$
- ❖ n^3 : $O(n^3)$
- ❖ n^2 : $O(n^2)$
- ❖ $n \log_2 n$: $O(n \log_2 n)$
- ❖ n : $O(n)$
- ❖ $\log_2 n$: $O(\log_2 n)$
- ❖ Hằng số : $O(c)$



Bài tập

❖ Tính Big-O của các hàm số sau:

$$1/ f(n) = 5n^4 + 3n^3 + 2n^2 + 4n + 1$$

$$6/ f(n) = 2n + 100\log n$$

$$2/ f(n) = 5n^2 + 3n\log n + 2n + 5$$

$$7/ f(n) = 2n$$

(ghi chú: $\log n = \log_2 n$)

$$8/ f(n) = 10$$

$$3/ f(n) = 20n^3 + 10n\log n + 5$$

$$4/ f(n) = 3\log n + 2$$

$$5/ f(n) = 2^{n+2}$$

Bài tập

❖ Tính độ phức tạp của thuật toán sau:

```
sum = 0;
```

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

```
    System.out.println(sum);
```

```
    sum = sum + i;
```

```
}
```

Bài tập

❖ Tính độ phức tạp của thuật toán sau:

Bước 1: Gán tổng = 0. Gán $i = 0$.

Bước 2:

- Tăng i thêm 1 đơn vị
- Gán tổng = tổng + i

Bước 3: so sánh i với 10

- Nếu $i < 10$, quay lại bước 2
- Ngược lại, nếu $i \geq 10$, dừng thuật toán

Gợi ý: số phép gán của thuật toán là bao nhiêu ? Số phép so sánh là bao nhiêu ?

Bài tập

❖ Tính độ phức tạp của thuật toán tính tổng dãy số sau:

$$S = 1 + \frac{1}{2} + \frac{1}{6} + \dots + \frac{1}{n!}$$

❖ Cho mảng A có chứa các số nguyên, có n phần tử, giải thuật sau xác định mảng A có chứa số nguyên x hay không

i = 0;

while (i < n && A[i] != x)

i++;

Tính độ phức tạp của giải thuật trên

Bài tập

□ Cho biết độ phức tạp của các thuật toán sau:

Thuật toán 1: `for (i=0; i<n; i++)`
 `for (j=0; j<n; j++)`
 `k++;`

Thuật toán 2: `for (i=0; i<n; i++)`
 `k++;`
 `for (i=0; i<n; i++)`
 `for (j=0; j<n; j++)`
 `k++;`

Thuật toán 3: `for (int i=0; i<n-1; i++)`
 `for (int j=0; j<i; j++)`
 `k += 1;`

Bài tập

□ Cho biết độ phức tạp của thuật toán sau:

```
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;  
}
```

Bài tập

□ Cho biết độ phức tạp của thuật toán sau:

```
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;  
}
```

Bài tập

□ Cho biết độ phức tạp của thuật toán sau:

```
sum = 0;
```

```
for (j = 0; j < n; j++)
```

```
    for (k = 0; k < n*n; k++)
```

```
        sum++;
```


Hỏi & Giải đáp



*"Formal education will make you a living;
self-education will make you a fortune"*

Bài học kế tiếp

1. Bài toán tìm kiếm

- ❖ Tìm kiếm tuần tự
- ❖ Tìm kiếm nhị phân

2. Bài toán sắp xếp

- ❖ Giải thuật sắp xếp cơ bản
- ❖ Giải thuật sắp xếp cải tiến

Tài liệu tham khảo

➤ Tài liệu môn học

- [1] Michael T. Goodrich, Roberto Tamassia, Data Structures & Algorithms in Java (6th Edition)
- [2] Trần Hạnh Nhi, Dương Anh Đức, Cấu trúc dữ liệu & giải thuật, Khoa CNTT, trường ĐH KHTN ĐHQG TpHCM

➤ Tài liệu tham khảo thêm

- [3] Thomas H. Cormen et al., 2009, Introduction to Algorithms, 3rd Edition, ebook.
- [4] Hoàng M. L., 2002, Cấu trúc dữ liệu và giải thuật, ĐHSP Hà Nội.