TRƯỜNG ĐẠI HỌC MỞ TP. HCM
Cơ hội học tập cho mọi người
Khoa Công Nghệ Thông Tin

Chương 1
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

GV: Lê Ngọc Hiếu

TP.HCM - Tháng 06 -2019

Mở đầu

Kiến thức cần thiết khi tìm hiểu về CTDL & GT:

- Dữ liệu/ thông tin là gì?
- Kiểu dữ liệu cơ bản, dữ liệu lưu trữ trong máy tính.
- Các kiến thức về cơ sở lập trình & kỹ thuật lập trình.

Mục tiêu dạy học

- Hiểu được khái niệm về "cấu trúc dữ liệu" và các ứng dụng.
- Hiểu được khái niệm về "giải thuật"
- Biết cách biểu diễn giải thuật.
- Biết cách đánh giá được độ phức tạp một giải thuật bằng công cụ Big O.

Nội dung chính

1.1 Cấu trúc dữ liệu

1.2 Giải thuật

1.3 Độ phức tạp của giải thuật

1.4 Tổng kết chương

1.5 Bài tập chương 1

Tài liệu tham khảo

1.1
CẤU TRÚC DỮ LIỆU
(Data Structure)

1.1 – CẤU TRÚC DỮ LIỆU

- CTDL là cấu trúc (sự tổ chức) của dữ liệu/thông tin lên trên máy tính, mà ở đó với cấu trúc này *máy tính có thể xử lý được*.
- Cấu trúc này phải rõ ràng, xác định, các thành phần bên trong cấu trúc cũng phải rõ ràng, và xác định.

1.1 – CẤU TRÚC DỮ LIỆU

Ví dụ 1.1: Cấu trúc dữ liệu *cơ bản* của một sinh viên

(mã số sv, họ và tên, giới tính, ngày sinh, địa chỉ)

<u>**Trong đó:**</u>

- mã số sinh viên, họ và tên, địa chỉ có kiểu dữ liệu là **kiểu chuỗi**.
- Ngày sinh của sinh viên có **kiểu Date** (kiểu ngày).

1.1 – CẤU TRÚC DỮ LIỆU

Ví dụ 1.1: Cấu trúc dữ liệu *cơ bản* của một sinh viên

(mã số sv, họ và tên, giới tính, ngày sinh, địa chỉ)

Trong đó:

- mã số sinh viên, họ và tên, địa chỉ có kiểu dữ liệu là **kiểu chuỗi**.
- Ngày sinh của sinh viên có **kiểu Date** (kiểu ngày).

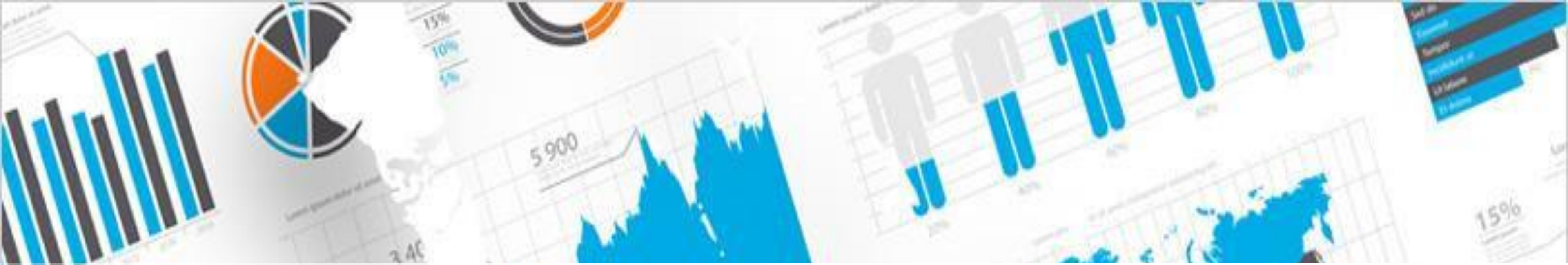
1.1 – CẤU TRÚC DỮ LIỆU

Ví dụ 1.2: Cấu trúc dữ liệu *cơ bản* của một lớp học

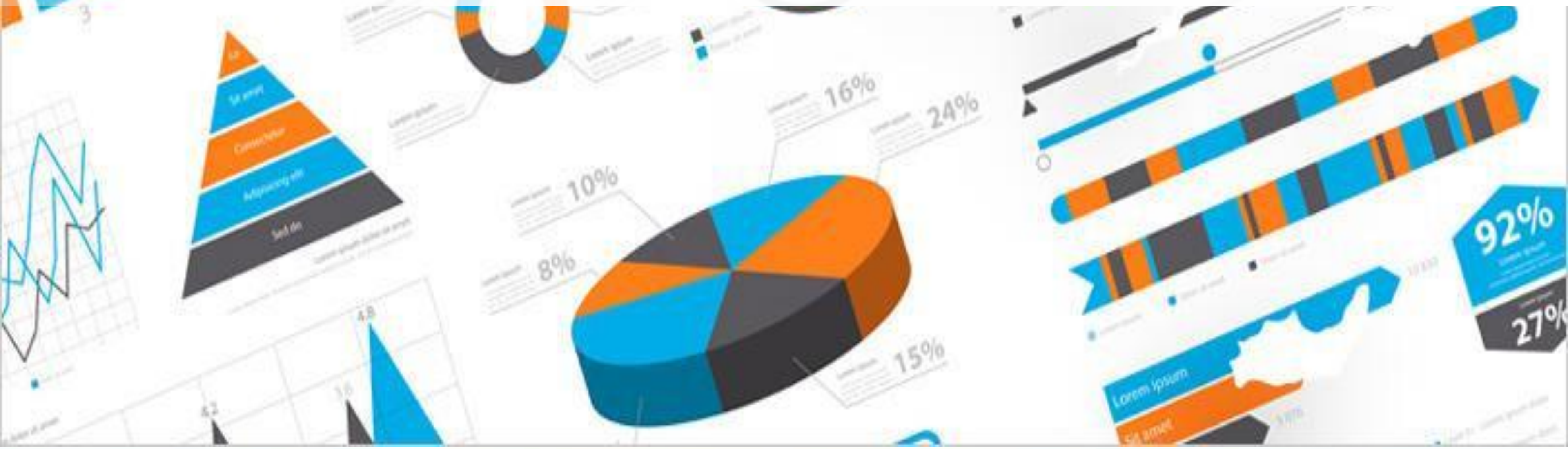
(Mã lớp, Tên lớp, tập sinh viên)

Trong đó:

- Mã lớp, tên lớp có kiểu dữ liệu là **kiểu chuỗi**.
- Tập sinh viên có **kiểu tập hợp** (tập hợp mà mỗi phần tử là một sinh viên)



1.2
GIẢI THUẬT (Algorithm)



1.2 – GIẢI THUẬT

- Giải thuật là một *tập hữu hạn* của các bước (chỉ thị hay hành động) theo một trình tự, được *xác định rõ ràng* nhằm mục đích để *giải quyết một bài toán* nào đó (dựa vào những giá trị đầu vào gọi là "***input***" và cho ra kết quả đầu ra gọi là "***ouput***")

1.2 – GIẢI THUẬT
Ví dụ 1.3

Trong kiến thức **Toán trung học cơ sở**, ta có bài toán: Tìm nghiệm phương trình bậc hai một ẩn có dạng $ax^2 + bx + c = 0$ (với: a, b, c $\in \mathbb{R}$; a $\neq$ 0).

*** Ta có **giải thuật (T)** để giải bài toán tìm nghiệm cho phương trình $ax^2 + bx + c = 0$ như sau:

Giải Thuật (T):

Đầu vào (**input**): a, b, c (a, b, c, $\in \mathbb{R}$)

Đầu ra (**output**): kết luận nghiệm

1.2 – GIẢI THUẬT
Ví dụ 1.3 (tiếp theo)

Bước 1: tính delta = $b^2 - 4ac$

Bước 2: thực hiện kiểm tra delta

2.1 Nếu delta < 0 thì

phương trình vô nghiệm;

2.2 Nếu delta = 0 thì

phương trình có nghiệm kép: $x_1 = x_2 = \frac{-b}{2a}$

2.3 Nếu delta > 0 thì

phương trình có hai nghiệm phân biệt:

$$x_1 = \frac{-b - \sqrt{delta}}{2a}$$

$$x_2 = \frac{-b + \sqrt{delta}}{2a}$$

1.2 – GIẢI THUẬT
Ví dụ 1.3 (tiếp theo)

NHẬN XÉT:

- (T) có số lượng bước giải hữu hạn (đếm được): *bước 1*, *bước 2.1*, *bước 2.2*, *bước 2.3*
- Các bước trong (T) rõ ràng, và có thể cài đặt trên máy tính được.
- (T) Nếu thực hiện theo đúng quy trình các bước (dựa vào giá trị a, b, c xác định "input") ta sẽ có kết luận về nghiệm (output)
- (T) Luôn cho kết quả đúng với bất kì giá trị a, b, c nào (a, b, c $\in \mathbb{R}$)

1.2 – GIẢI THUẬT
Ví dụ 1.3 (tiếp theo)

Cài đặt trên C++

```
void TimNghiem(float a, float b, float c)
{
     float delta = b*b – 4*a*c, x1, x2;
     if (delta<0)
         cout<<"Phuong trinh vo nghiem";
     else if (delta==0)
     {
         x1 =-b/(2*a);
         x2 =-b/(2*a);
         cout<<"Phuong trinh co Nghiem kep x1 ="<<x1<<" x2 = "<<x2;
    }
    else if (delta>0)
    {
          x1 = (-b-sqrt(delta))/(2*a);
          x2 = (-b+sqrt(delta))/(2*a);
        cout<<"Phuong trinh co 2 Nghiem kep x1 = "<<x1<<"  x2 = "<<x2;
    }
}
```

1.2 – GIẢI THUẬT

GIẢI THUẬT ĐÚNG

📖 Giải Thuật đúng là giải thuật sẽ dừng lại với kết quả đúng (cho ra kết quả đúng) mọi trường hợp của đầu vào (theo bài toán).

Ví dụ 1.4:

TH1: a = 1, b = -2, c = 1 thì TimNghiem(a,b,c)

=> cho ra kết quả đúng; (x1 = 1, x2 = 1);

TH2: a = 1, b = 3, giải c = 2 thì TimNghiem(a,b,c)

=> cho ra kết quả đúng; (x1 = -2, x2 = -1);

….

THn: a = .. b = .., c = .. thì TimNghiem(a,b,c)

=> cho ra kết quả đúng; (…);

1.2 – GIẢI THUẬT

GIẢI THUẬT SAI

📖 Giải thuật sai là giải thuật nếu tồn tại một trường hợp đầu vào khiến cho giải thuật không dừng hoặc dừng với một kết quả không đúng (hoặc không phù hợp).

Ví dụ 1.5: giả sử (T) bỏ đi bước 2.3 (delta > 0) thì chắc chắn (T) sẽ *không cho ra kết quả gì* với trường hợp *có hai nghiệm phân biệt*. (vì (T) đã xét thiếu trường hợp này).

TH1: a = 1, b = -2, c = 1 thì TimNghiem(a,b,c)

=> cho ra kết quả đúng; (x1 = 1, x2 = 1);

TH2: a = 1, b = 3, c = 2 thì TimNghiem(a,b,c)

=> không cho ra kết quả (*không đúng*);

MỘT SỐ TÍNH CHẤT CỦA GIẢI THUẬT

- Tính đúng: giải thuật cho ra **kết quả** quả đúng từ các ***đầu vào*** tương ứng.
- Tính dừng: giải thuật phải dừng ở một hữu hạn bước (không lặp vô hạn).
- Tính rõ ràng, xác định: Các bước trong thuật toán phải tường minh (không mập mờ, không ẩn bên trong các thao tác con).
- Tính khách quan: giải thuật phải độc lập với ngôn ngữ lập trình, có thể được viết bằng các ngôn ngữ lập trình khác nhau, bởi nhiều người khác nhau, nhưng cho ra kết quả giống nhau.

1.2 – GIẢI THUẬT

MỘT SỐ PHƯƠNG PHÁP BIỂU DIỄN GT

Ta có 03 cách cơ bản để biểu diễn, mô tả giải thuật:

- Ngôn ngữ tự nhiên
- Lưu đồ
- Mã giả

1.2 – GIẢI THUẬT

MỘT SỐ PHƯƠNG PHÁP BIỂU DIỄN GT

NGÔN NGỮ TỰ NHIÊN

- Là một dạng trình bày giải thuật dựa hoàn toàn bằng ngôn ngữ tự nhiên.
- Phải đảm bảo được các tiêu chuẩn về giải thuật
  - Tính đúng
  - Tính dừng
  - Tính rõ ràng, xác định.

1.2 – GIẢI THUẬT

MỘT SỐ PHƯƠNG PHÁP BIỂU DIỄN GT

Ví dụ 1.6: Biểu diễn bằng ngôn ngữ tự nhiên: giải phương trình $ax^2 + bx + c = 0$ $(a \neq 0)$

Cho a,b,c $(a \neq 0)$

Xuất: nghiệm phương trình

Tính $\Delta = b^2 - 4ac$;

Nếu $\Delta < 0$ thì phương trình vô nghiệm;

Nếu $\Delta = 0$ thì phương trình có nghiệm kép: $x_1 = x_2 = \frac{-b}{2a}$;

Nếu $\Delta > 0$ thì phương trình có hai nghiệm phân biệt:

$x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ và $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$;

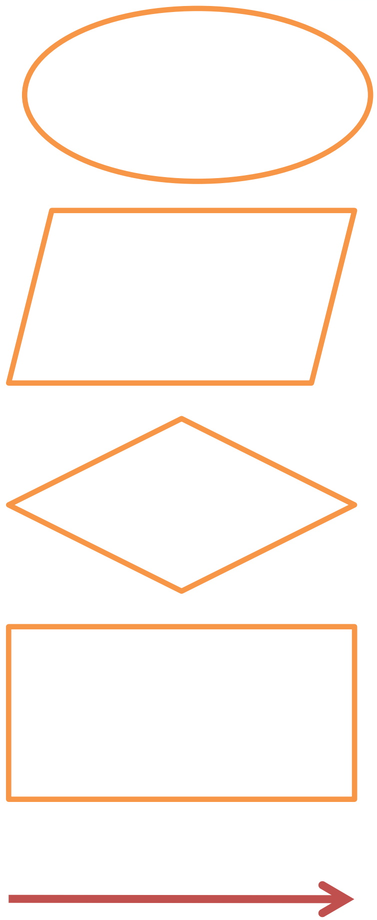
MỘT SỐ PHƯƠNG PHÁP BIỂU DIỄN GT

LƯU ĐỒ

- Là dạng trình bày giải thuật theo các quy ước chuẩn.
- Là bộ quy ước chung của các lập trình viên, các nhà phân tích thiết kế giải thuật.

1.2 – GIẢI THUẬT
MỘT SỐ PHƯƠNG PHÁP BIỂU DIỄN GT

Ý NGHĨA CÁC KÍ HIỆU Trong LƯU ĐỒ



Khối giới hạn
Chỉ thị bắt đầu và kết thúc.

Khối vào ra
Nhập/Xuất dữ liệu.

Khối lựa chọn
Tùy điều kiện sẽ rẽ nhánh.

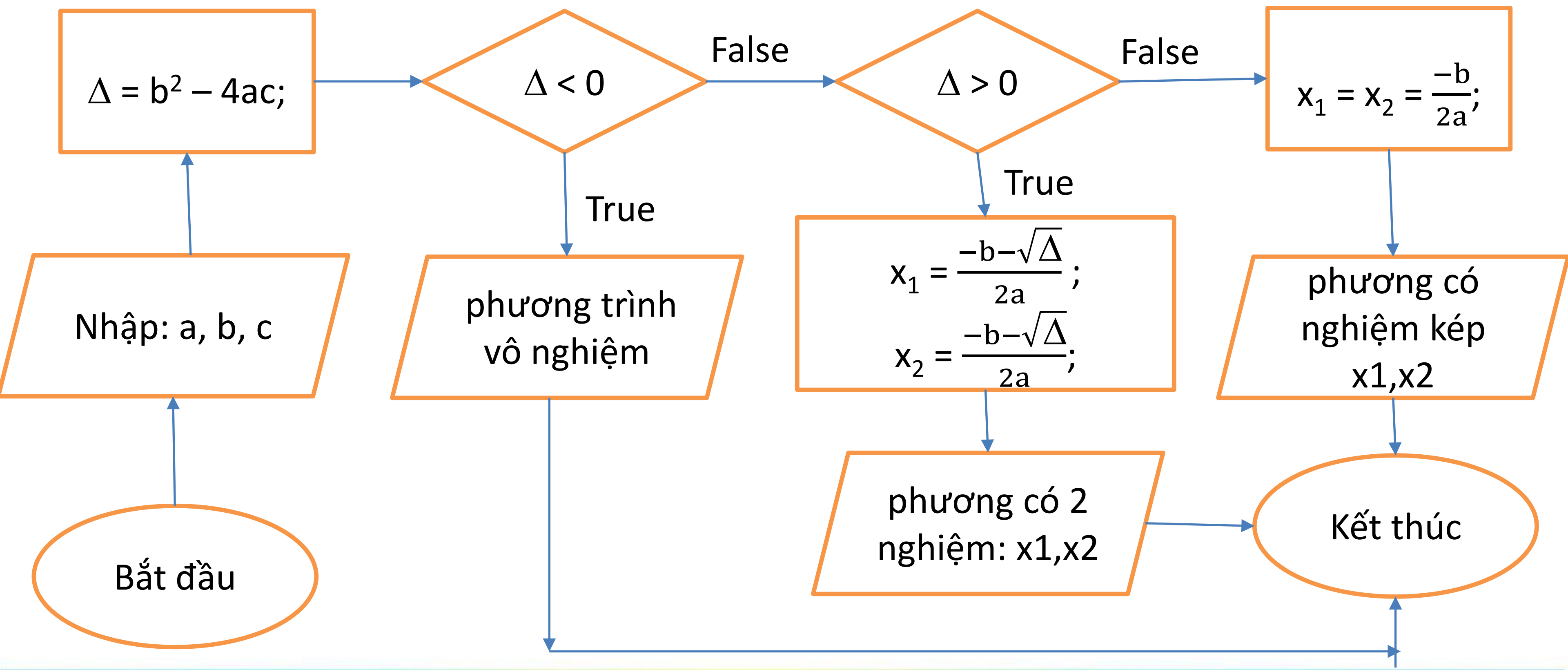
Khối thao tác
Ghi thao tác cần thực hiện.

Đường đi
Chỉ hướng thao tác tiếp theo.

1.2 – GIẢI THUẬT

MỘT SỐ PHƯƠNG PHÁP BIỂU DIỄN GT

Ví dụ 1.7: Biểu diễn bằng lưu đồ:
giải phương trình $ax^2 + bx + c = 0$ $(a \neq 0)$



MỘT SỐ PHƯƠNG PHÁP BIỂU DIỄN GT

MÃ GIẢ

- Là dạng ngôn ngữ quy ước tự nhiên kết hợp với các quy ước toán học hoặc ngôn ngữ lập trình

Ví dụ 1.8: Biểu diễn bằng mã giả:
giải phương trình $ax^2 + bx + c = 0$ $(a \neq 0)$

1.2 – GIẢI THUẬT

MỘT SỐ PHƯƠNG PHÁP BIỂU DIỄN GT

```
void TimNghiem(a, b, c)
{
    delta = b*b – 4*a*c, x1, x2;
    Nếu delta<0
        thì phương trình vô nghiệm;
    Nếu delta==0
        phương trình có hai nghiệm phân biệt:
        x1 =-b/(2*a);
        x2 =-b/(2*a);
    Nếu delta>0 thì:
      Phuong trinh co 2 Nghiem kep x1
      x1 = (-b-sqrt(delta))/(2*a);
      x2 = (-b+sqrt(delta))/(2*a);
}
```

1.3

ĐỘ PHỨC TẠP CỦA GIẢI THUẬT

1.3 – Độ phức tạp của giải thuật

- Giải thuật được đưa ra để giải quyết một bài toán nào đó?
- Vấn đề nếu có 2 hoặc nhiều hơn 2 giải thuật cùng giải quyết một bài toán thì ta chọn giải thuật nào?

Đọc dễ hiểu, Ít vùng nhớ, Ngôn ngữ LT, ...?

Time $\Leftrightarrow$ Độ phức tạp (time)

$\Rightarrow$ Làm thế nào để đo được độ phức tạp???

1.3 – Độ phức tạp của giải thuật

- Ước lượng thời gian chạy của một giải dựa vào kích cỡ đầu vào.

Ví dụ 1.9: Cho một mảng số nguyên gồm n phần tử, hãy kiểm tra x có tồn tại trong mảng hay không? -> n phần tử (cỡ n).

Ví dụ 1.10: Thực hiện sắp xếp một mảng số nguyên gồm n phần tử theo thứ tự tăng dần. -> n phần tử (cỡ n)

1.3 – Độ phức tạp của giải thuật

Xét lại ví dụ 1.9:

n = 10 (phần tử)

4	3	2	6	8	7	10	1	9	5

X = 4 — $\Rightarrow$ **1 lần** — **Trường hợp tốt nhất**

X = 5 — $\Rightarrow$ **10 lần** — **Trường hợp xấu nhất**

1.3 – Độ phức tạp của giải thuật

CÁC TRƯỜNG HỢP ĐÁNH GIÁ

- Trường hợp tốt nhất
- Trường hợp trung bình
- Trường hợp xấu nhất

1.3 – Độ phức tạp của giải thuật

MỘT SỐ PHƯƠNG PHÁP ĐÁNH GIÁ

- Accounting Method (***Phương pháp đếm***): đếm các phép toán cơ bản bên trong của một thuật toán.
- Potential Method
  [5 or https://en.wikipedia.org/wiki/Potential_method].
- Dynamic Table
  [https://www.cs.cornell.edu/courses/cs3110/2009sp/lectures/lec21.html]

1.3 – Độ phức tạp của giải thuật PHƯƠNG PHÁP ĐẾM

PHƯƠNG PHÁP ĐẾM - Accounting Method

- Đếm các phép toán cơ bản bên trong một thuật toán.

Các phép toán số học: +, -, *, /..

Các phép toán so sánh: $<$, $>$, $\geq$, $\leq$, …

Các phép gán, …

T(n)

1.3 – Độ phức tạp của giải thuật PHƯƠNG PHÁP ĐẾM

NGUYÊN TẮC

- Độ phức tạp về thời gian của một thuật toán được xác định bằng số lượng các thao tác cơ bản cần thiết để giải quyết vấn đề đặt ra.

1.3 – Độ phức tạp của giải thuật PHƯƠNG PHÁP ĐẾM

MỤC TIÊU ĐÁNH GIÁ

- Xác định thời gian chạy của thuật toán là một hàm theo kích thước của dữ liệu đầu vào.
- Xác định xem số lượng các thao tác cơ bản phụ thuộc vào kích thước input như thế nào :
  - n : kích thước đầu vào (input)
  - T(n): số các thác tác cơ bản

1.3 – Độ phức tạp của giải thuật PHƯƠNG PHÁP ĐẾM

ƯỚC LƯỢNG TIỆM CẬN

- O : Big Oh
- Ω : Big Omega
- θ : Big Theta
- o : Little Oh
- ω : Little Omega

O : Big Oh

1.3 – Độ phức tạp của giải thuật BIG OH

ĐỊNH NGHĨA BIG O

Giả sử cho T(n) là hàm có tốc độ thời gian theo tăng n như sau:

$$T(n) = 4n^2 - 2n + 2$$

Nếu ta bỏ qua các hằng số và các n có hệ số lũy thừa thấp hơn (hay còn gọi là tốc độ tăng thấp hơn) thì ta có thể nói "*T(n) có tộc độ tăng theo n^2*", và ta viết như sau:

$$T(n) \approx O(n^2) \text{ hay } T(n) = O(n^2)$$

$\Rightarrow$ Ta đọc là độ phức tạp T(n) thuộc lớp $O(n^2)$

1.3 – Độ phức tạp của giải thuật BIG OH

ĐỊNH NGHĨA BIG O

- Cho f(n) và g(n) là hai hàm số thực, ta nói

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

Nếu và chỉ nếu tồn tại một hằng số C, K sao cho:

$$|f(n)| \leq C*|g(n)|, \forall n > K$$

Có nghĩa là tốc độ tăng của f(n) nhỏ hơn g(n).

1.3 – Độ phức tạp của giải thuật BIG OH

Ví dụ 1.11: Cho $f(n) = 5n^2 + 2n + 6$ (n là số nguyên dương)

$\Leftrightarrow f(n) \leq 5n^2 + 2n^2 + 6n^2$

$\Leftrightarrow f(n) \leq 13n^2$

Đặt $C = 13$ và chọn $k = 1$

$\Leftrightarrow f(n) \approx O(n^2)$ hay $f(n) = O(n^2)$

$\Rightarrow$ Ta nói f(n) có độ phức tạp thuộc lớp $O(n^2)$

1.3 – Độ phức tạp của giải thuật BIG OH

CÁC BƯỚC ĐÁNH GIÁ ĐỘ PHỨC TẠP THUẬT TOÁN

- Xác định các thao tác cơ bản của một thuật toán.
- Thực hiện tính tổng (đếm) các thao tác cơ bản T(n)
- Kiểm tra thuộc lớp nào của Big O.

1.3 – Độ phức tạp của giải thuật BIG OH

(1) Cách tính các thao tác cơ bản vòng lặp **for**

```
int TongTu1DenN(int n)
{
    int sum = 0;
    for(int i = 1;  i <= n;  i++)
        sum = sum + i;
    return sum;
}
```

1.3 – Độ phức tạp của giải thuật BIG OH

Phép gán 'gan':

Lần 0 thì 'gan' thực hiện 1 + 1 = 2 lần

Lần 1 thì 'gan' thực hiện 1 + 3 = 4 lần

……

Lần k thì 'gan' thực hiện 1 + (2k+1) lần

Lần n thì 'gan' thực hiện 1 + (2n+1) lần

$T(n) = 1 + (2n+1) = 2n+2 \Rightarrow \mathbf{T(n) \approx O(2n+2) \approx O(n)}$

1.3 – Độ phức tạp của giải thuật BIG OH

Phép so sánh ‘so_sanh’:

Lần 0 thì ‘so_sanh’ thực hiện 1 lần

Lần 1 thì ‘so_sanh’ thực hiện 2 lần

Lần 2 thì ‘so_sanh’ thực hiện 3 lần

……

Lần k thì ‘so_sanh’ thực hiện k+1 lần

Lần n thì ‘so_sanh’ thực hiện n+1 lần

O(n)

1.3 – Độ phức tạp của giải thuật BIG OH

Nhận xét về vòng lặp for

- Thời gian thực thi một vòng lặp for tối đa bằng thời gian thực thi các phép toán cơ bản (bên trong for) nhân với số lượng vòng lặp;
- Độ phức tạp của vòng lặp for thuộc lớp:

$$O(n)$$

Với n là kích cỡ đầu vào.

1.3 – Độ phức tạp của giải thuật BIG OH

(2) Cách tính các thao tác cơ bản vòng **for lồng nhau**

```
int TinhTongMaTran(int a[][], int n)
{
    int sum = 0;
    for(int i = 0;  i < n;  i++)
        for(int j = 0;  j < n;  j++)
            sum = sum + a[i][j];
    return sum;
}
```

1.3 – Độ phức tạp của giải thuật BIG OH

i = 0, j chạy từ 0 đến n-1 = n lần chạy

i = 1, j chạy từ 0 đến n-1 = n lần chạy

i = 2, j chạy từ 0 đến n-1 = n lần chạy

……

i = n-2, j chạy từ 0 đến n-1 = n lần chạy

i = n-1, j chạy từ 0 đến n-1 = n lần chạy

$T(n) = n * n \approx O(n^2)$

$\Rightarrow T(n) \approx$ **$O(n^2)$**

1.3 – Độ phức tạp của giải thuật BIG OH

Nhận xét vòng lặp for lồng nhau

- Vòng lặp for lồng nhau: thời gian thực thi vòng lặp for lồng nhau bằng thời gian thực thi các phép toán cơ bản nhân với tích kích thước của mỗi vòng lặp.

1.3 – Độ phức tạp của giải thuật BIG OH

(3) Cách tính các thao tác cơ bản của các **đoạn chương trình kế tiếp nhau** (nối tiếp)

```
void main(int n)
{
    int sum1, sum2, n, a[20][20];
    cout<<"Nhap n: ";
    cin>>n;
    NhapMaTran(a, n);              ⇔ O(n^2)
    sum1 = TongTu1DenN(n);         ⇔ O(n)
    sum2 = TongMaTran(a, n);       ⇔ O(n^2)
}
```

$T(n) = \mathbf{max}(O(n^2), O(n), O(n^2))$

1.3 – Độ phức tạp của giải thuật BIG OH

(4) Cách tính các thao tác cơ bản của câu lệnh điều kiện **if…else**

```
If <condition>
    S1;
else
    S2;
```

Độ phức tạp của chương trình là độ phức tạp lớn nhất của S1 và S2

T(n) = **max**(do_phuc_tap(S1), do_phuc_tap(S2));

1.3 – Độ phức tạp của giải thuật BIG OH

(5) Đánh giá giải thuật **đệ quy**

```
int TinhTong(int n)
{
  if(n ==1)
     return 1;
  return n + TinhTong(n-1);
}
```

$$\begin{cases} T(n) = C1 \ (\text{khi } n = 1) \\ T(n) = T(n-1) + C2 \ (n > 1) \end{cases}$$

$T(n) = T(n-1) + C2$

$\Leftrightarrow (T(n-2) + C2) + C2 = T(n - 2) + 2C2$

$\Leftrightarrow (T(n-3) + C2) + 2C2 = T(n-3) + 3C2$

……

$\Leftrightarrow (T(n-k) + C2) + (k-1)C2 = T(n-k) + kC2$

Chương trình dừng khi $n - k = 1 \Rightarrow k = n-1$

$\Leftrightarrow T(n-n-1) + (n-1)C2 = T(1) + (n-1)C2$

$\Leftrightarrow C1 + (n-1)C2$

$T(n) = C1 + nC2 - C2 \approx$ **$O(n)$**

1.3 – Độ phức tạp của giải thuật BIG OH

Một số chú ý khi đánh giá giải thuật **đệ quy**

- Xác định được công thức đệ quy
- Giải công thức đệ quy

1.3 – Độ phức tạp của giải thuật BIG OH

Ví dụ 1.12: Phân tích độ phức tạp của giải thuật Insertion Sort

```
void InsertionSort(int a[], int n)
{
    int i, j, x;
    for (i = 1;  i < n;  i++)
    {
        x = a[i]; j=i;
        while (j > 0 && a[j-1] > x)
        {
            a[j] = a[j-1];
            j- -;
        }
        a[j] = x;
    }
}
```

1.3 – Độ phức tạp của giải thuật BIG OH

Gọi **a[j-1] > x (*)** là phép toán cơ bản

(n-1)

Với **i = 1** thì j chạy 1 lần => (*) **1** chạy lần
Với **i = 2** thì j chạy 2 lần => (*) **2** chạy lần
Với **i = 3** thì j chạy 3 lần => (*) **3** chạy lần
......
Với **i = n-2** thì j chạy n-2 lần => (*) chạy **n-2** lần
Với **i = n-1** thì j chạy n-1 lần => (*) chạy **n-1** lần

$T(n) = (n-1)+(n-2)+(n-3) + ... + 2 + 1$

$= n^2/2 + n/2 \approx O(n^2)$

1.3 – Độ phức tạp của giải thuật BIG OH

Ví dụ 1.14: đánh giá độ phức tạp của thuật toán sau:

```
bool TimX(int a[], int n, int x)
{
    int mid, left = 0, right = n;
    while (left<=right)
     {
        mid = (left+right)/2;
        if(a[mid] = = x)              return true;
        else if(a[mid]<x)             right = mid-1;
        else                          left = mid + 1;
    }
    return false;
}
```

1.3 – Độ phức tạp của giải thuật BIG OH

$$\begin{cases} T(n) = 3C1 \ (\text{khi } n = 1) \\ T(n) = T(n/2) + 3C2 \ (n > 1) \end{cases}$$

T(n) = T(n/2) + 3C2

⇔ (T(n/4) +3C2) + 3C2 = T(n/4) + 6C2

⇔ (T(n/8) + 3C2) + 6C2 = T(n/8) + 9C2

……

⇔ $(T(n/2^k) + 3C2) + (k\text{-}1)3C2 = T(n/2^k) + 3kC2$

Chương trình dừng khi $n/2^k = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

⇔ $T(1) + 3\log_2 nC2 = 3C1 + 3\log_2 nC2$

$T(n) = 3C1 + 3\log_2 nC2 \approx$ **$O(\log_2 n)$**

1.3 – Độ phức tạp của giải thuật BIG OH

MỘT SỐ THUẬT NGỮ DÙNG TRONG

ĐỘ PHỨC TẠP THƯỜNG GẶP

Độ phức tạp hằng số	Độ phức tạp Lô ga rít	Độ phức tạp tuyến tính	Độ phức tạp nlogn	Độ phức tạp đa thức	Độ phức tạp hàm mũ	Độ phức tạp giai thừa
O(1)	**O(logn)**	**O(n)**	**O(nlogn)**	**O(n^k)**	**O(k^n)** **k > 1**	**O (n!)**

→

Độ phức tạp tăng dần

1.4 – TỔNG KẾT
CHƯƠNG 1

1.4 – Tổng kết chương 1

- Ý niệm về “Cấu trúc dữ liệu” và “giải thuật”
- Một số phương pháp biểu diễn giải thuật
- Cách đánh giá độ phức tạp giải thuật dựa trên ước lượng tiệm cận Big Oh (Ô lớn)

1.4 – Tổng kết chương 1

1.5- Bài tập rèn luyện
CHƯƠNG 1

1.5 - Bài tập chương 1

CÂU HỎI

- **Câu 1:** Trong khoa học máy tính, ***cấu trúc dữ liệu*** được hiểu như thế nào? Cho ví dụ.
- **Câu 2:** Trong khoa học máy tính, ***giải thuật*** được hiểu như thế nào? Cho ví dụ.
- **Câu 3:** Tại sao nói CTDL và GT có quan hệ mật thiết với nhau? Liệt kê 1 ví dụ nói về cách thiết kế cấu trúc dữ liệu sẽ ảnh hưởng đến giải thuật, giải thích tại sao?
- **Câu 4:** Đếm số ***phép so sánh*** trong giải thuật ở ví dụ 1.12.

1.5 - Bài tập chương 1

- **Bài 1:** Đếm số phép toán gán, phép so sánh được thực thi và xác định độ phức tạp trong đoạn code sau:

```
for (i = 0; i < n; i++)
   for (j = 0; j < m; j++)
         if (a[ i ][ j ] = = x) return 1;
return -1;
```

- **Bài 2:** Đếm số phép toán gán, phép so sánh được thực thi và xác định độ phức tạp trong đoạn code sau:

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

1.5 - Bài tập chương 1

Bài 3: Đánh giá độ phức tạp của đoạn code sau:

```
for (i = 0; i < n; i++)
        sum1+=i;
for (i = 0; i < n*n; i++)
        sum2+=i;
```

Bài 4: Đánh giá độ phức tạp của hàm tính giai thừa sau:

```
int GT(int n)
{
        if (n == 1)
                return 1;
        return n*GT(n-1);
}
```

1.5 - Bài tập chương 1

Bài 5: Đánh giá độ phức tạp của hàm tính dãy FIBONACCI sau:

```
int Fibo(int n)
{
    if (n <=1)
        return n;
    return Fibo(n-1) + Fibo(n-2);
}
```

Hướng dẫn

- Tất cả sinh viên phải trả lời các câu hỏi, làm bài và nộp qua LMS của trường.
- Bài tập chương 1 làm trên file WORD; trong bài làm ghi rõ họ tên, lớp, bài tập chương và các thông tin cần thiết.
- Khuyến khích sử dụng tiếng Anh trong bài tập.

⇒**Ngày nộp:** trước khi học chương 3.

⇒**Cách nộp:** sử dụng ***github*** để nộp bài, sau đó nộp lên LMS của trường.

Tài liệu tham khảo

- **Dương Anh Đức**, *Giáo trình cấu trúc dữ liệu & giải thuật* (*Chương 1)*, 2010, ĐH KHTN TP.HCM
- **Thomas H.Cormen, Charles E.Leiserson, Ronald L. Rivest, Cliffrod Stein**, *(Chapter 10) Introduction to Algorithms*, Third Edition, 2009.
- **Adam Drozdek**, *(Chapter 3) Data Structures and Algorithms in C*++, Fourth Edtion, CENGAGE Learning, 2013.

Phụ lục – Thuật ngữ tiếng Anh

#	Tiếng Anh	Phiên Âm	Tiếng Việt
1	Data	/ ‘deɪtə /	Dữ liệu
2	Structure	/ ‘strʌktʃə(r) /	Cấu trúc
3	Algorithm	/ ‘ælgərɪðəm /	Thuật toán / Thuật giải / Giải thuật
4	Complexity	/ kəm’pleksəti /	Độ phức tạp

KẾT THÚC CHƯƠNG 1

Trường Đại học Mở TP.HCM

Khoa Công Nghệ Thông Tin