CHƯƠNG 1:
GIỚI THIỆU THUẬT GIẢI

GVGD: Ths NGUYỄN CHÍ THANH
Email: thanh.nc@ou.edu.com

MỤC TIÊU

- Hiểu được khái niệm về "cấu trúc dữ liệu" và các ứng dụng.
- Hiểu được khái niệm về "giải thuật"
- Hiểu được *mối quan hệ* giữa cấu trúc dữ liệu và giải thuật
- Biết cách biểu diễn giải thuật.
- Biết cách tính/đánh giá được độ phức tạp một thuật giải, bằng phương pháp đếm.

NỘI DUNG

- Một số khái niệm cơ bản
- Một số phương pháp biểu diễn thuật giải
- Độ phức tạp của thuật giải

1.1 CẤU TRÚC DỮ LIỆU (Data struture)

- 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.

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 là Date (kiểu ngày).

Ví dụ 1.2:

Cấu trúc dữ liệu *cơ bản* của một lớp học

(Mã lớp, Tên lớp, tập các sinh viên)

<u>Trong đó:</u> Mã lớp, tên lớp có kiểu dữ liệu là kiểu chuỗi.

Tập các sinh viên có kiểu là một tập hợp, (mỗi phần tử có kiểu dữ liệu là một sinh viên)

1.2 THUẬT GIẢI (Algorithms)

- Thuật giải 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***")

Ví dụ 1.3: trong kiến thức **Toán trung học cơ sở**, ta có bài toán về "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 thuật giải (T) để giải bài toán tìm nghiệm cho phương trình $ax^2 + bx + c = 0$ như sau:

Thuật giải (T):

Đầu vào (**input**): a, b, c ($a, b, c, \in \mathbb{R}$)
Đầu ra (**output**): kết luận nghiệm

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:

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

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

- (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 $\in \mathbb{R}$)

Thuật giải (T):

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

Đầu ra (**output**): kết luận nghiệm

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:

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

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

```
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";
    if (delta==0)
    {
        x1 =-b/(2*a);
        x2 =-b/(2*a);
        cout<<"Phuong trinh co Nghiem kep x1 =
        "<<x1<<" x2 = "<<x2;
    }
    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;
    }
}
```

Thuật giải (T):

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

Đầu ra (output): kết luận nghiệm

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:

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

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

NHƯ THẾ NÀO LÀ THUẬT GIẢI ĐÚNG

- Thuật giải đúng là thuật giải 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, 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; (…);

THẾ NÀO LÀ THUẬT GIẢI SAI

- Thuật giải sai là thuật giải nếu tồn tại một trường hợp đầu vào khiến cho thuật giải 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 THUẬT GIẢI

- Tính đúng: Thuật giải cho ra **kết quả** quả đúng từ các ***đầu vào*** tương ứng.
- Tính dừng: Thuật giải 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: Thuật giải 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.

MỘT SỐ PHƯƠNG PHÁP BIỂU DIỄN THUẬT GIẢI

- Ngôn ngữ tự nhiên
- Lưu đồ (sơ đồ khối)
- Mã giả (Pseudocode)

NGÔN NGỮ TỰ NHIÊN

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

Ví dụ 1.6: 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: $x_1 = x_2 = \frac{-b}{2a}$;

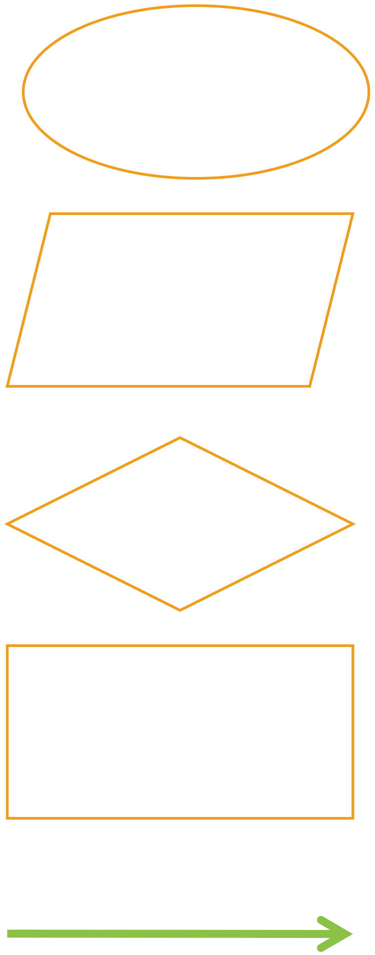
Nếu $\Delta > 0$ thì phương trình có hai nghiệm phân biệt:

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

SƠ ĐỒ KHỐI

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

Ý NGHĨA CÁC KÍ HIỆU



Khối giới hạn
Chỉ thị bắt đầu và kết thúc.

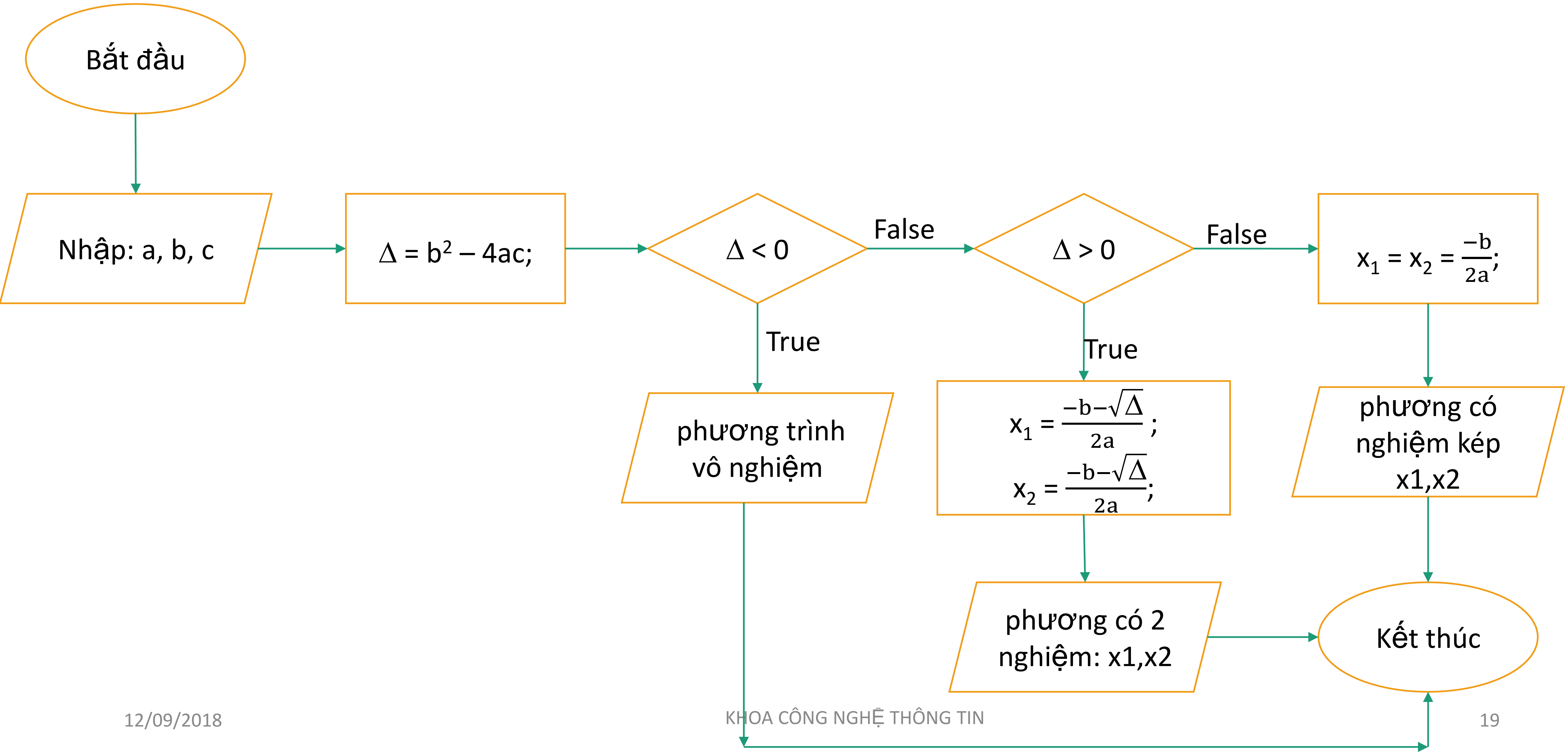
Khối vào ra
Nhập/Xuất dữ liệu.

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.

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



MÃ GIẢ (Pseudocode)

- Là loại lai giữa ngôn ngữ lập trình và ngôn ngữ tự nhiên.
- Hoặc là là dạng một ngôn ngữ quy ước (theo chuẩn).
- Hoặc là dạng ngôn ngữ lập trình.

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

```
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);

}
```

MỐI QUAN HỆ GIỮA CTDL VÀ THUẬT GIẢI

Ví dụ 1.9

STT	English	Tiếng Việt
1	Animal	Động vật
2	Award	Giải thưởng
3	Apple	Quả táo
4	Bread	Bánh mì
5	Busy	Bận rộn
6	Bus	Xe buýt
7	Chair	Cái ghế
8	City	Thàn phố
9	Dog	Con chó
10	Design	Thiết kế

Tra từ “Dog”

Dog **⇒ 9 lần**

NHẬN XÉT

- Phải dò tuần tự từ vị trí thứ 1 đến vị trí 10
- Nếu tìm thấy thì thực hiện lấy giá trị Tiếng Việt tương ứng, xuất ra ngoài màn hình.
- Cách này sẽ chạy lâu nếu số từ Tiếng Anh lưu trữ lớn.

MỐI QUAN HỆ GIỮA CTDL VÀ THUẬT GIẢI

STT	Index		English	Tiếng Việt
1	A	1	Animal	Động vật
		2	Award	Giải thưởng
		3	Apple	Quả táo
2	B	1	Bread	Bánh mì
		2	Busy	Bận rộn
		3	Bus	Xe buýt
3	C	1	Chair	Cái ghế
		2	City	Thàn phố
4	D	1	Dog	Con chó
		2	Design	Thiết kế

Tra từ "Dog"

D $\Rightarrow$ **4 lần**

Dog $\Rightarrow$ **1 lần**

Tổng = 5 lần

NHẬN XÉT

- Không tổ chức tuần tự danh mục từ tiếng Anh. Mà thực hiện nhóm các từ Tiếng Anh có cùng chữ cái đầu tiên lại với nhau.
- Khi nhập vào một từ Tiếng Anh cần tìm, Chỉ cần tìm chữ cái đầu tiên của từ này với nhóm chữ cái đầu tiên được lưu trữ.
- Nếu tìm thấy thì thực hiện dò từng từ trong “nhóm từ” vừa tìm được.
- Nếu tìm thấy trong nhóm này thi thực hiện xuất nghĩa Tiếng việt tương ứng.
- Cách này sẽ chạy tốt hơn cách lưu trữ tuần tự nếu số từ Tiếng Anh lưu trữ lớn. (vẫn chưa phải là cách tốt).

MỐI QUAN HỆ GIỮA CTDL VÀ THUẬT GIẢI

- Cấu dữ liệu khác nhau, sẽ có những xử lý đặc thù khác nhau (⇒ thuật giải khác nhau).
- Cấu trúc dữ liệu giống nhau, nhưng trạng thái của dữ liệu khác nhau thì cũng sẽ có những xử lý đặc thù riêng.
- Thuật giải và cấu trúc dữ liệu có quan hệ mật thiết với nhau (không tách rời).
- Khi thiết kế cấu trúc dữ liệu cho một hệ thống có nghĩa là bạn cần phải nghĩ đến các thao tác (hay vấn đề) trên cấu trúc dữ liệu này để thuật tiện cho phần thiết kế thuật giải.

- Thuật giải được đưa ra để giải quyết một bài toán nào đó?

- Vấn đề nếu có 2 hoặc nhiều hơn 2 thuật giải/thuật toán cùng giải quyết một bài toán thì ta chọn thuật giải nào?

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

Time ⇔ Độ phức tạp (time)

⇒ Làm thế nào để đo được độ phức tạp???

1.3. ĐỘ PHỨC TẠP THUẬT GIẢI

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

Ví dụ 1.10: 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 1.11: Thực hiện sắp xếp một mảng số nguyên gồm n phần tử theo thứ tự tăng dần. -> n phần tử (cỡ n)

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

CÁC TRƯỜNG HỢP ĐÁNH GIÁ

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

MỘT SỐ PHƯƠNG PHÁP ĐÁNH GIÁ

- Accounting Method 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]

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: <, >, ≥, ≤, …

  Các phép gán, …

T(n)

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.

MỤC TIÊU ĐÁNH GIÁ

- Xác định thời gian chạy của thuật toán là một hàm theo kích thước của dữ liệu nhập.
- 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

ƯỚC LƯỢNG TIỆM CẬN (Asymptotic Notations)

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

O : Big Oh

ĐỊNH NGHĨA 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 $O(n^2)$

(Formal) ĐỊNH NGHĨA 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).

Ví dụ 1.12: Cho $f(n) = 5n^2 + 2n + 6$ (n là số nguyên dương)

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

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

Bỏ qua hằng số C = 13 (không đáng kể)

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

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

CÁC BƯỚC ĐÁNH GIÁ ĐỘ PHỨC TẠP THUẬT TOÁN

- **Tìm/nhận diện 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) Cách tính các thao tác cơ bản vòng lặp for

Vòng lặp for

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

Phép gán ‘gan’:

n = 0 thì ‘gan’ thực hiện 1 + 1 = 2 lần

n = 1 thì ‘gan’ thực hiện 1 + 3 = 4 lần

n = 2 thì ‘gan’ thực hiện 1 + 5 = 6 lần

n = 3 thì ‘gan’ thực hiện 1 + 7 = 8 lần

……

n = k thì ‘gan’ thực hiện 1 + (2k+1) lần

n = 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) Cách tính các thao tác cơ bản vòng lặp for

📖 **Vòng lặp for**

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

Phép so sánh 'so_sanh':

n = 0 thì 'so_sanh' thực hiện 1 lần

n = 1 thì 'so_sanh' thực hiện 2 lần

n = 2 thì 'so_sanh' thực hiện 3 lần

n = 3 thì 'so_sanh' thực hiện 4 lần

……

n = k thì 'so_sanh' thực hiện k+1 lần

n = n thì 'so_sanh' thực hiện n+1 lần

Số phép so sánh < số phép phép gán O(n)

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.

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

🕮 Vòng lặp for lồng nhau

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

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

$$T(n) = 1 + 2 + \ldots + (n-2) + (n-1) \approx \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$

$$\approx O(n^2 + n) \approx O(n^2) \quad \Rightarrow T(n) \approx O(n^2)$$

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

🕮 Vòng lặp for lồng nhau

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

n lần

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

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

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

Nhận xét vòng lặp for lồng nhau

- <u>**Vòng lặp for lồng nhau:**</u> **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.**

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

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

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

```
void main(int n)
{
    int sum1, sum2, n, a[20][20];
    cout<<"Nhap n: ";
    cin>>n;
    NhapMaTran(a, n);            ⇔ O(n²)
    sum1 = TongTu1DenN(n);       ⇔ O(n)
    sum2 = TongMaTran(a, n);     ⇔ O(n²)
}
```

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

(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));

(5) Đánh giá thuật giải 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)**

Một số chú ý khi đánh giá thuật giải đệ quy

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

Ví dụ 1.13: Phân tích độ phức tạp của thuật giải 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;
    }
}
```

(n-1)

i = 1, j chạy từ 1 về 1 = 0 lần chạy ⇒ 1 + 3 + 1'gan' + 0 * 2'gan'

i = 2, j chạy từ 2 về 1 = 1 lần chạy ⇒ 1 + 6 + 2 'gan' + 1 * 2'gan'

i = 3, j chạy từ 3 về 1 = 2 lần chạy ⇒ 1 + 9 + 3 'gan' + 2 * 2'gan'

i = 4, j chạy từ 4 về 1 = 3 lần chạy ⇒ 1 + 12 + 4 'gan' + 3 * 2'gan'

....

i = n-2, j chạy từ n-2 về 1 = n-3 lần chạy ⇒ 1 + 3*(n-2) + (n-2) 'gan' + n-3 * 2'gan'

i = n-1, j chạy từ n-1 về 1 = n-2 lần chạy ⇒ 1 + 3*(n-1) + n-1 'gan' + (n-2) * 2'gan'

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

$$\Leftrightarrow T(n) = (n-1) * (6n - 5) = 6n^2 - 11n + 5$$

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

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;
}
```

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

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

$\Leftrightarrow (T(n/4) + 3C2) + 3C2 = T(n/4) + 6C2$

$\Leftrightarrow (T(n/8) + 3C2) + 6C2 = T(n/8) + 9C2$

......

$\Leftrightarrow (T(n/2^k) + 3C2) + (k-1)3C2 = T(n/2^k) + 3kC2$

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

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

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

TỔNG KẾT CHƯƠNG

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

TÀI LIỆU THAM KHẢO

1. **Thomas H.Cormen**, **Charles E.Leiserson**, **Ronald L. Rivest**, **Cliffrod Stein**, (Chapter 10) *Introduction to Algorithms*, Third Edition, 2009.

2. **Adam Drozdek**, (Chapter 3) *Data Structures and Algorithms in C++*, Fourth Edtion, CENGAGE Learning, 2013.

Bài tập chương 1

- Bài 1: Liệt kê 1 ví dụ nói về cách thiết kế cấu trúc dữ liệu sẽ ảnh hưởng đến thuật giải, giải thích tại sao?
- Bài 2: Đếm số phép toán ***so sánh*** trong thuật giải ở ví dụ 1.13

- Bài 3: Đếm số phép toán gán, phép toán 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 4:Đếm số phép toán gán, phép toán 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++;
```

Bài 5: 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 6: 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);
}
```

📖 * Bài 7: 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);
}
```