

Tên bài giảng

Mở đầu

Môn học: **Phân tích thiết kế thuật toán**
Chương: 1
Hệ: Đại học
Giảng viên: TS. Phạm Đình Phong
Email: phongpd@utc.edu.vn
ĐT: 0972481813

Nội dung bài học

1. **Giới thiệu môn học**
2. **Thuật toán**
3. **Phân tích thiết kế thuật toán**
4. **Đánh giá độ phức tạp thuật toán**

Giới thiệu môn học

- Mục tiêu về kiến thức
 - Người học **giải thích được** khái niệm thuật toán, đánh giá, so sánh được **độ phức tạp** của các thuật toán và các phương pháp thiết kế thuật toán
 - **So sánh** những **điểm mạnh yếu** của các phương pháp thiết kế thuật toán: chia để trị, quay lui, tham lam, quy hoạch động khi áp dụng vào từng bài toán, từ đó lựa chọn cấu trúc dữ liệu và thuật toán phù hợp cho bài toán cụ thể
 - **Vận dụng** các phương pháp phân tích thiết kế thuật toán vào lập trình giải các bài toán cụ thể

Giới thiệu môn học

- Mục tiêu về kỹ năng
 - **Diễn giải** được vai trò, vị trí của thuật toán trong xây dựng một ứng dụng phần mềm
 - **Trình bày** được sự thỏa hiệp giữa các mục tiêu khi thiết kế các thuật toán
 - **Suy xét, lựa chọn** các thuật toán cho các bài toán thực tế

Giới thiệu môn học

- Mục tiêu về thái độ
 - Người học ngày càng yêu thích ngành học, có thái độ học tập tích cực, sáng tạo
 - Có ý thức vận dụng các kiến thức được học vào giải quyết các bài toán thực tế bằng công nghệ thông tin

Giới thiệu môn học

- Mục tiêu về nhận thức
 - Người học thấy được vai trò của thuật toán trong công nghệ thông tin, đánh giá so sánh được độ phức tạp của thuật toán để xây dựng những thuật toán tốt nhất có thể cho những bài toán cụ thể
 - Người học vận dụng các cấu trúc dữ liệu trong lập trình, gắn liền với thuật toán như Stack, Queue, List, Tree, Set, Map, ...
 - Người học có tư duy phân tích xây dựng các thuật toán khác nhau giải quyết các bài toán

Giới thiệu môn học

- Các học phần tiên quyết
 - Cấu trúc dữ liệu và giải thuật
 - Toán rời rạc
- Phân bổ giờ tín chỉ đối với các hoạt động
 - Lý thuyết: 30 tiết
 - Bài tập: 15 tiết
 - Thực hành: 15 tiết
 - Tự học: 90 tiết

Giới thiệu môn học

- Đánh giá kết quả học tập học phần
 - Chuyên cần, đánh giá thường xuyên: 10%
 - Thực hành: 30%
 - Hình thức: Thảo luận, thực hành
 - Đánh giá kết thúc học phần:
 - Tỷ trọng: 60%
 - Hình thức: Viết hoặc báo cáo bài tập lớn

Giới thiệu môn học

- Tài liệu tham khảo
 - Giải thuật và lập trình, Lê Minh Hoàng, Trường ĐHSPHN
 - An Introduction to Algorithms, Thomash H. Cormen, Charles E. Leiserson, Ronald L. Rivest (Biên dịch: Ngọc Anh Thư Press), Nhà xuất bản Thống Kê, 2001
 - Cẩm nang thuật toán, Vol 1+2, Robert Sedgetwic Nhà xuất bản khoa học kỹ thuật

Thuật toán

- Khái niệm thuật toán
 - Một dãy hữu hạn các thao tác và trình tự thực hiện các thao tác đó sao cho sau khi thực hiện dãy thao tác này theo trình tự đã chỉ ra với đầu vào xác định ta thu được kết quả đầu ra mong muốn
 - Một chương trình là một cách biểu diễn hình thức thuật toán bởi một ngôn ngữ lập trình cụ thể

Thuật toán

- Khái niệm thuật toán

- Ví dụ: Thuật toán Euclid tìm ước chung lớn nhất của hai số nguyên a, b

Theo nhà toán học Hy Lạp cổ đại Euclid thì ước chung lớn nhất của hai số nguyên a, b (ký hiệu (a, b)) bằng ước chung lớn nhất của b với phần dư a chia cho b ($(a, b) = (b, a \bmod b)$). Do đó, ta dùng phép lặp biến đổi tới khi b bằng 0 và thu được ước chung lớn nhất là a .

Thuật toán

- Khái niệm thuật toán
 - Ví dụ: Thuật toán Euclid tìm ước chung lớn nhất của hai số nguyên a, b

```
int Euclid(int a, int b) {  
    while (b) {  
        int r = a % b;  
        a = b;  
        b = r;  
    }  
    return a;  
}  
  
int main() {  
    int x, y;  
    input(x); input(y);  
    printf("Uoc chung lon nhat la %d", Euclid(x, y));  
}
```

Thuật toán

- Các đặc trưng của thuật toán
 - **Đầu vào**
 - **Đầu ra**
 - **Tính đúng đắn**: đảm bảo kết quả tính toán hay các thao tác mà máy tính thực hiện được là chính xác
 - **Tính xác định**: các bước thực hiện có trình tự xác định
 - **Tính dừng**: phải cho ra kết quả sau một số hữu hạn bước
 - **Tính phổ dụng**: áp dụng cho các bài toán cùng dạng
 - **Tính khách quan**: cho kết quả như nhau khi chạy trên các máy tính khác nhau

Thuật toán

- Các đặc trưng của thuật toán
 - **Đầu vào** và **Đầu ra**
 - Mỗi thuật toán được gắn với một bài toán cụ thể cần xác định **đầu vào** và **đầu ra** và **điều kiện** của chúng
 - Ví dụ: bài toán tìm ước số chung lớn nhất thì
 - Đầu vào: a và b
 - Đầu ra: ước số chung lớn nhất của a và b
 - Điều kiện: a và b là hai số nguyên, đầu ra cũng là một số nguyên

Thuật toán

- Các đặc trưng của thuật toán
 - **Tính đúng đắn:** kết quả của thuật toán là chính xác
 - **Ví dụ:** Nhập vào các số thực a, b, c và biện luận về nghiệm của phương trình $ax^2 + bx + c = 0$
 - **Phân tích thuật toán:**
 - Nếu $a = 0$ phương trình suy biến về $bx + c = 0$
 - Nếu $b = 0$ và $c = 0$ thì phương trình có vô số nghiệm
 - Nếu $b = 0$ và $c \neq 0$ thì phương trình vô nghiệm
 - Nếu $b \neq 0$ thì nghiệm $x = -c/b$
 - Nếu a khác 0, biện luận theo $\Delta = b^2 - 4ac$
 - Nếu $\Delta < 0$ thì phương trình vô nghiệm
 - Nếu $\Delta = 0$ thì phương trình có nghiệm kép $x = -b/(2a)$
 - Nếu $\Delta > 0$ thì phương trình có hai nghiệm phân biệt:
 $x_1 = (-b - \sqrt{\Delta})/(2a)$ và $x_2 = (-b + \sqrt{\Delta})/(2a)$

Thuật toán

- **Tính đúng đắn:** kết quả của thuật toán là chính xác
- **Ví dụ:** Nhập vào các số thực a, b, c và biện luận về nghiệm của phương trình $ax^2 + bx + c = 0$

```
#include<math.h>
int main() {
    float a, b, c, delta;
    scanf("%f%f%f", &a, &b, &c);
    if(a==0) {
        if(b == 0) printf(c ? "vo nghiem":"vo so nghiem");
        else printf("%.3f", -c/b);
    } else {
        delta=b*b-4*a*c;
        if (delta<0) printf("vo nghiem");
        else if (delta == 0) printf("%.2f", -b/(2*a));
        else {
            delta = sqrt(delta);
            printf("%.3f\n%.3f", (-b-delta)/(2*a), (-b+delta)/(2*a));
        }
    }
}
```


Thuật toán

- Các đặc trưng của thuật toán
 - **Tính xác định (rõ ràng):** Thuật toán phải được thể hiện bằng các câu lệnh tường minh, các câu lệnh được sắp xếp theo thứ tự (trình tự) nhất định (bước nào trước, bước nào sau)
- ❑ Bài toán: Hãy tìm ƯSCLN của hai số nguyên dương a, b .
 - Input: a, b hai số nguyên
 - Output: ƯSCLN của a và b là một số nguyên
- ❑ Thuật toán
 - Bước 1: Tìm số dư r của phép chia a cho b
 - Bước 2: Kiểm tra:
 - Nếu $r = 0$ thì thông báo b là ƯSCLN kết thúc
 - Nếu $r \neq 0$ thì thực hiện đặt $a=b, b=r$ rồi quay lại và làm lại Bước 1

Thuật toán

- Các đặc trưng của thuật toán
 - **Tính dừng:** Thuật toán chỉ có ý nghĩa nếu nó dừng sau hữu hạn bước, còn chạy vô hạn không dừng thì không thu được kết quả
 - **Ví dụ:** Hãy chỉ ra tính dừng của thuật toán tìm kiếm tuần tự

Bước 1. Nhập N , các số hạng $a_1 a_2 \dots a_N$ và khoá k ;

Bước 2. $i = 1$;

Bước 3. Nếu $a_i = k$ thì thông báo chỉ số i và kết thúc;

Bước 4. $i = i + 1$;

Bước 5. Nếu $i > N$ thì thông báo dãy A không có số hạng nào có giá trị bằng k và kết thúc;

Bước 6. Quay lại bước 3;

Thuật toán

- Các đặc trưng của thuật toán

- **Tính dừng:** Thuật toán chỉ có ý nghĩa nếu nó dừng sau hữu hạn bước, còn chạy vô hạn không dừng thì không thu được kết quả
- **Ví dụ:** Hãy chỉ ra tính dừng của thuật toán tìm kiếm tuần tự

Tính dừng của thuật toán tìm kiếm tuần tự là thuật toán phải kết thúc sau một số hữu hạn lần bước tính.

Thuật toán chia làm hai trường hợp:

- Nếu tìm thấy giá trị cần tìm trong dãy A ($a_i = k$) thì thông báo chỉ số i (vị trí tìm thấy khoá k trong dãy A) và kết thúc.
- Nếu không tìm thấy giá trị cần tìm trong dãy A, vì bước 4 thực hiện việc tăng giá trị của i lớn hơn 1 nên sau N lần thì $i > N$, thông báo dãy A không có giá trị nào bằng k và kết thúc.

Thuật toán

- Các đặc trưng của thuật toán
 - **Tính phổ dụng:** Một tư tưởng thuật toán hóa có thể áp dụng giải nhiều bài toán trong một lớp các bài toán nào đó
 - **Ví dụ:** Thuật toán tìm kiếm nhị phân
Bài toán nhập vào dãy a_1, a_2, \dots, a_n đã được sắp xếp không giảm. Tìm vị trí của phần tử x nếu nó xuất hiện trong dãy.

Thuật toán

- Các đặc trưng của thuật toán

- **Ví dụ:** Thuật toán tìm kiếm nhị phân

- Thuật toán là tìm x trong dãy a từ vị trí L đến vị trí R , ban đầu $L = 1$, $R = n$
 - Cắt đôi đoạn $[L, R]$ bởi điểm giữa $M = (L + R)/2$ (phép chia nguyên)
 - Nếu $x = a_M$ thì ta chỉ ra vị trí là M và dừng
 - Nếu $x < a_M$ thì vì dãy đã sắp xếp tăng dần nên nếu có x trong dãy thì nó sẽ nằm trong đoạn $[L, M - 1]$ ta gán lại $R = M - 1$
 - Nếu $x > a_M$ thì gán lại $L = M + 1$ để thu nhỏ lại không gian tìm kiếm
 - Lặp lại bước lặp tới khi tìm thấy x hoặc tới khi không gian tìm kiếm rỗng ($L > R$) thì dừng

Thuật toán

- Biểu diễn thuật toán
 - Cách 1: Ngôn ngữ tự nhiên
 - Cách 2: Ngôn ngữ lưu đồ (sơ đồ khối)
 - Cách 3: Mã giả
 - Cách 4: Các ngôn ngữ lập trình như Pascal, C/C++, Java, ... nhưng không nhất thiết phải sử dụng đúng ký pháp của ngôn ngữ đó

Thuật toán

- Biểu diễn thuật toán bằng ngôn ngữ tự nhiên
 - Sử dụng ngôn ngữ thường ngày để liệt kê các bước của thuật toán
 - Đặc điểm:
 - + Không yêu cầu người viết thuật toán cũng như người đọc thuật toán phải nắm các quy tắc
 - + Cách biểu diễn này thường dài dòng, không thể hiện rõ cấu trúc của thuật toán, đôi lúc gây hiểu lầm hoặc khó hiểu cho người đọc
 - + Không có một quy tắc cố định nào trong việc thể hiện thuật toán bằng ngôn ngữ tự nhiên. Nên viết các bước con lùì vào bên phải và đánh số bước theo quy tắc phân cấp như 1., 1.1., 1.1.1., ...

Ví dụ giải bài toán tính giai thừa

Bước 1: Đọc dữ liệu vào là số nguyên n

Bước 2: Kiểm tra

- Nếu $n = 0$ thì in ra giá trị giai thừa bằng 1 rồi kết thúc
- Còn ngược lại thì thực hiện các bước tiếp theo.

Bước 3:

- Đặt $\text{GiaiThua} = 1$,
- Đặt $i = 1$,

Bước 4: Kiểm tra nếu i vẫn còn $\leq n$ thì đi lặp lại bước 5 rồi đến bước 6 cho đến khi $i > n$ thì dừng

Bước 5: Tính $\text{GiaiThua} = \text{GiaiThua} * i$ (Tính $\text{GiaiThua} * i$ rồi lại nhớ giá trị biểu thức này vào GiaiThua)

Bước 6: Tăng i lên 1 đơn vị





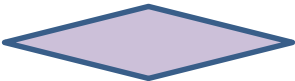


Bước 7: In ra giá trị GiaiThua rồi kết thúc

Sử dụng lưu đồ thuật toán

- Thuật toán được mô tả dưới dạng sơ đồ khối
- Sử dụng các ký hiệu khác nhau để biểu thị chuỗi hoạt động, cần thiết để giải quyết một vấn đề
- Đóng vai trò như một bản thiết kế hoặc sơ đồ logic của giải pháp cho một vấn đề

Lưu đồ thuật toán

□ Các ký hiệu trong lưu đồ thuật toán

Kí hiệu	Ý nghĩa	Chú thích
	Biểu diễn điểm bắt đầu hay kết thúc của chương trình	
	Biểu diễn các thao tác nhập/xuất	
	Biểu diễn các thao tác xử lý hoặc tính toán	
	Biểu diễn lời gọi chương trình con	
	Quyết định rẽ nhánh theo điều kiện	
	Điểm kết nối	
	Đường kết nối chỉ trình tự thao tác	

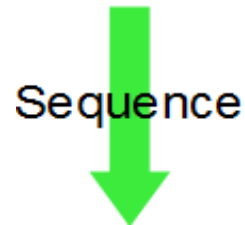
Các cấu trúc logic

- ❑ Cấu trúc tuần tự
- ❑ Cấu trúc rẽ nhánh

- if
- if ... else
- switch ... case

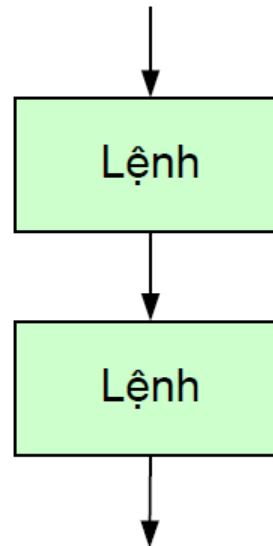
- ❑ Cấu trúc lặp

- while
- do ... while
- for



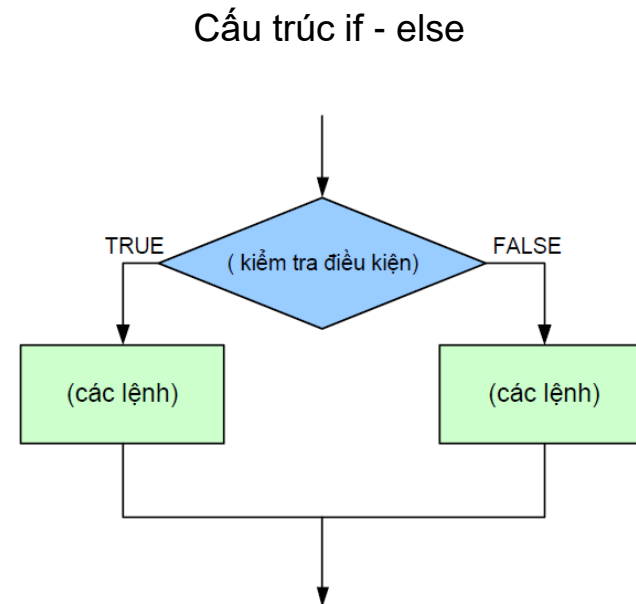
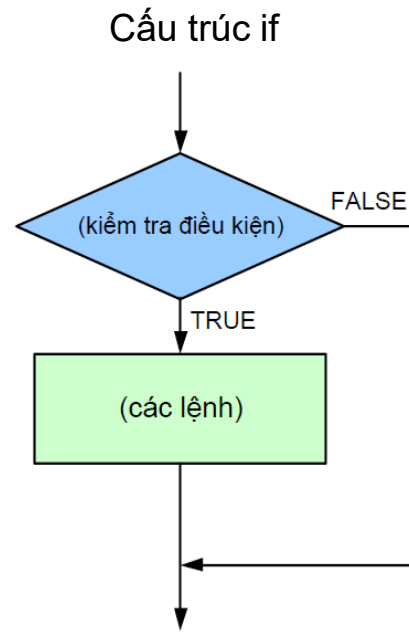
Cấu trúc tuần tự

- Các lệnh được thực hiện tuần tự theo chiều mũi tên



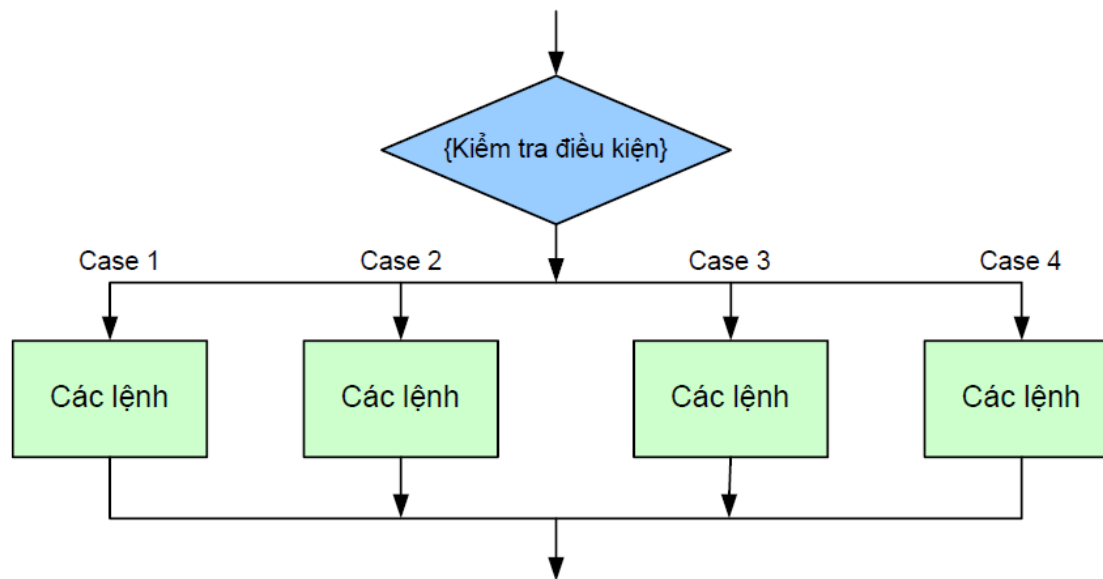
Cấu trúc rẽ nhánh

- ❑ Kiểm tra điều kiện đúng sẽ thực hiện theo nhánh TRUE
- ❑ Kiểm tra điều kiện sai sẽ thực hiện theo nhánh FALSE



Cấu trúc rẽ nhiều nhánh switch ... case

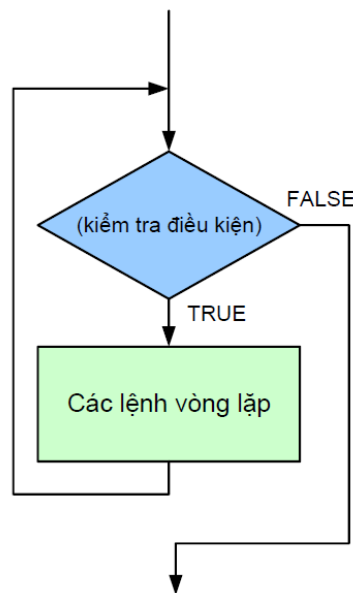
- ❑ Giá trị của biểu thức rơi vào trường hợp nào thì thực hiện theo nhánh đó



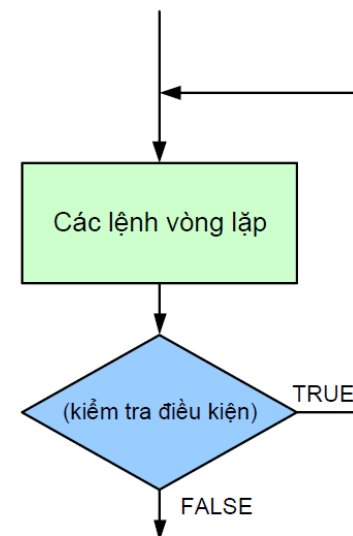
Cấu trúc lặp không biết trước số lần lặp

- ❑ Thực hiện lặp đi lặp lại một số lệnh nào đó khi điều kiện lặp vẫn còn đúng. Nói cách khác, điều kiện lặp sai thì dừng lặp
- ❑ while: kiểm tra điều kiện lặp trước
- ❑ do ... while: kiểm tra điều kiện lặp sau

Cấu trúc while



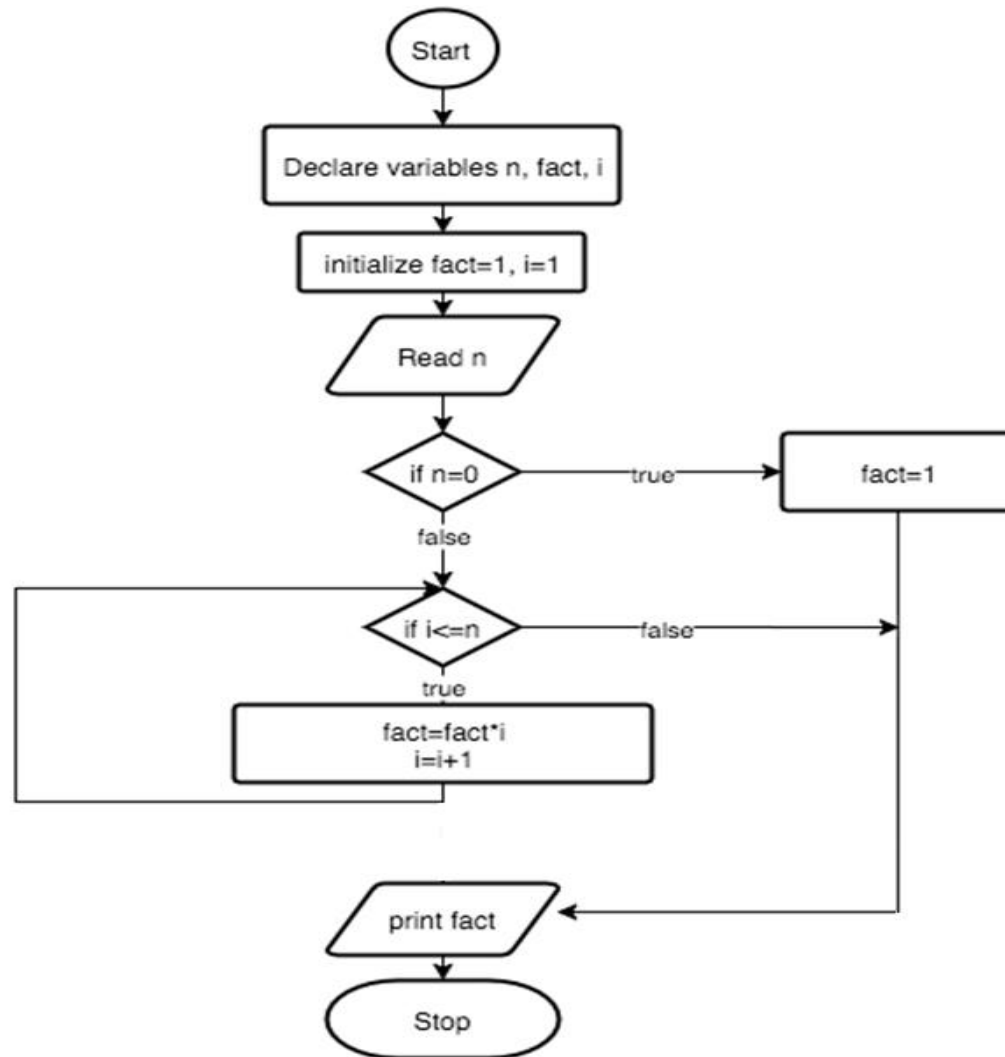
Cấu trúc do ... while



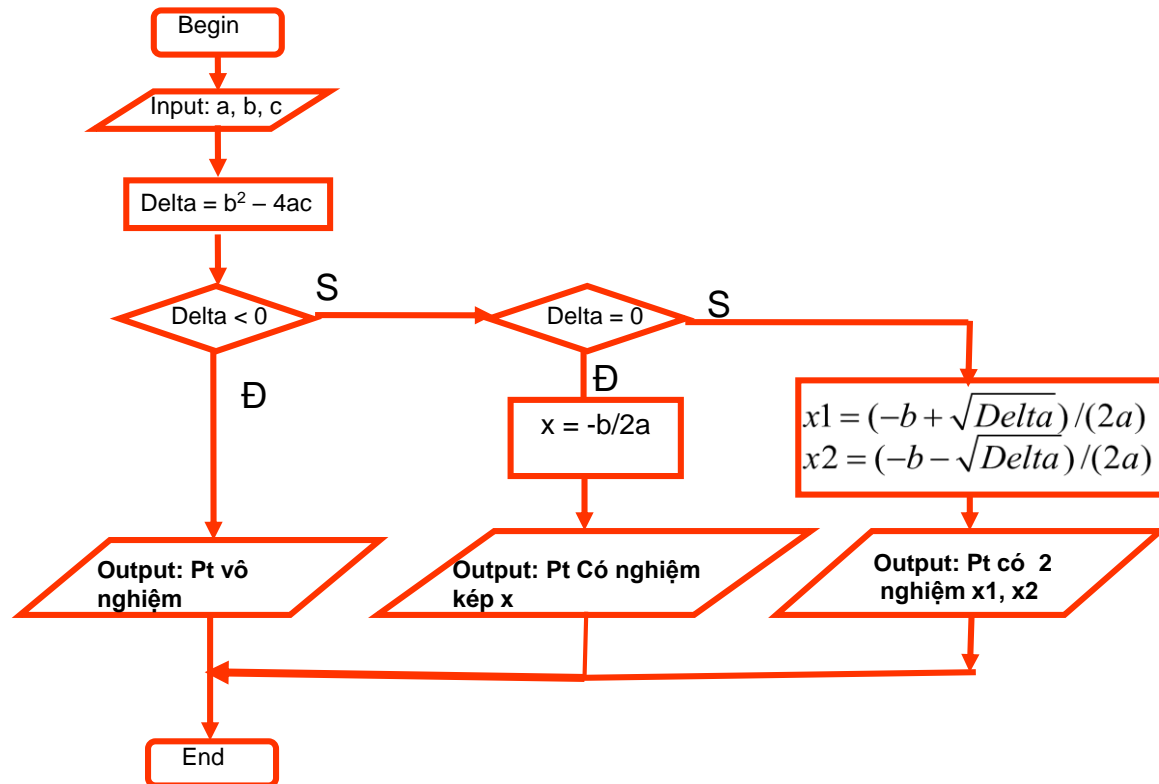
Cấu trúc lặp biết trước số lần lặp

- ❑ Cấu trúc **for**
- ❑ Các lệnh của vòng lặp được thực hiện với số lần lặp biết trước
- ❑ Có thể coi cấu trúc **for** tương đương với cấu trúc **while** hoặc **do ... while**

Ví dụ lưu đồ thuật toán tính giai thừa

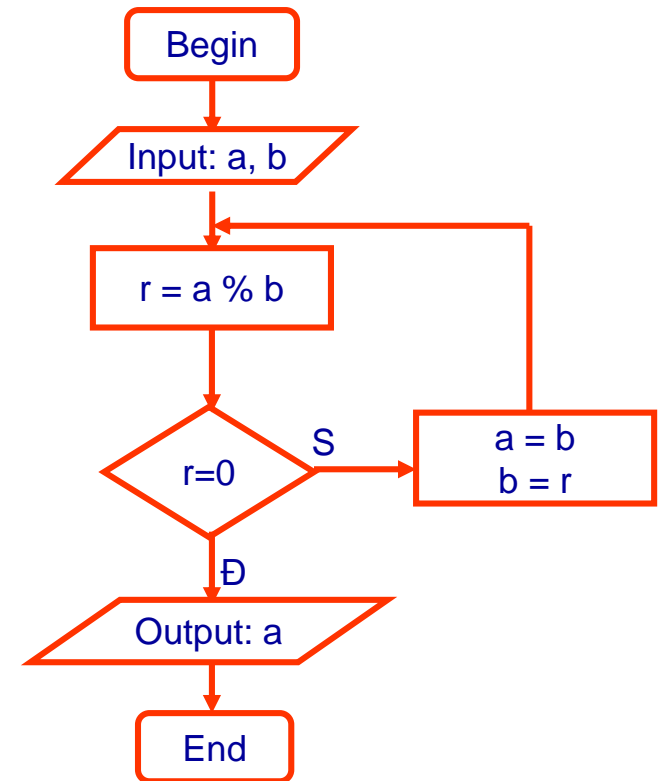


Ví dụ lưu đồ thuật toán giải phương trình bậc hai



Ví dụ lưu đồ thuật toán tìm ƯSCLN của hai số nguyên

- Bài tập: Mô tả các bước thực hiện theo lưu đồ thuật toán để tìm ƯSCLN của (35,75), (256,68)



Sử dụng mã giả (pseudo code)

- Khi thể hiện thuật toán bằng mã giả, ta sẽ vay mượn các cú pháp của một ngôn ngữ lập trình nào đó để thể hiện thuật toán
- Dùng mã giả vừa tận dụng được các khái niệm trong ngôn ngữ lập trình, vừa giúp người cài đặt dễ dàng nắm bắt nội dung thuật toán
- Trong mã giả vẫn có thể dùng một phần ngôn ngữ tự nhiên

Ví dụ giải phương trình bậc hai

Sử dụng mã giả ngôn ngữ Pascal

$\text{delta} = b*b - 4*a*c$

if $\text{delta} > 0$ **then begin**

$x1 = (-b - \text{sqrt}(\text{delta})) / (2*a)$

$x2 = (-b + \text{sqrt}(\text{delta})) / (2*a)$

Xuất kết quả: phương trình có hai nghiệm là $x1$ và $x2$

end

else

if $\text{delta} = 0$ **then**

Xuất kết quả : phương trình có nghiệm kép là $-b/(2*a)$

else {trường hợp $\text{delta} < 0$ }

Xuất kết quả: phương trình vô nghiệm

Phân tích thuật toán

- Độ phức tạp của thuật toán
 - Độ phức tạp tính toán của thuật toán được xác định như là lượng tài nguyên các loại mà thuật toán đòi hỏi sử dụng
 - Có hai loại tài nguyên quan trọng đó là thời gian và bộ nhớ (không gian lưu trữ)
 - Việc tính chính xác được các loại tài nguyên mà thuật toán đòi hỏi là rất khó. Vì thế ta quan tâm đến việc đưa ra các đánh giá sát thực cho các đại lượng này
 - Trong môn học này ta đặc biệt quan tâm đến đánh giá thời gian cần thiết để thực hiện thuật toán mà ta sẽ gọi là **thời gian tính** của thuật toán

Phân tích thuật toán

- Phép toán cơ bản (phép tính cơ bản)
 - Đo thời gian tính bằng đơn vị đo nào?
 - **Định nghĩa.** Ta gọi ***phép toán cơ bản*** là phép toán có thể thực hiện với thời gian bị chặn bởi một hằng số không phụ thuộc vào kích thước dữ liệu đầu vào
 - Để tính toán thời gian tính của thuật toán ta sẽ đếm ***số phép toán cơ bản*** mà nó phải thực hiện
 - **Ví dụ:** trong phân tích thuật toán, ta thường coi các phép gán, cộng, trừ, nhân, chia, lũy thừa, ... là các phép toán cơ bản

Phân tích thuật toán

- Đếm số phép toán cơ bản
 - Nhận diện các phép toán cơ bản trong thuật toán
 - Nếu có nhiều phép toán cơ bản thì xác định phép toán cơ bản T chiếm nhiều thời gian chạy nhất so với các phép toán cơ bản còn lại (ví dụ: $i = i + 1 \rightarrow$ coi phép cộng chạy lâu hơn phép gán)
 - Phép toán T thường xuất hiện trong các vòng lặp
 - Đếm số lần thực hiện phép toán T, sẽ thu được hàm thời gian chạy $f(n)$

Phân tích thuật toán

- Ví dụ đếm số phép toán cơ bản

Ví dụ 1: In các phần tử
for (i = 0; i < n; i++)
 cout << a[i] << endl;

Số lần in ra màn hình = n

Ví dụ 2:
Nhân ma trận tam giác
dưới với véctơ (mã giả)
for (i = 0; i < n; i++)
 c[i] = 0
for (i = 0; i < n; i++)
 for (j = 0; j < i; j++)
 c[i] = c[i] + a[i][j] * b[j];

Số phép nhân: $n(n+1)/2$

Ví dụ 3: Kiểm tra tính sắp xếp

```
template <class T>
bool isSorted(T *a, int n)
{
    bool sorted = true;
    for (int i=1; i<n; i++)
        if (a[i-1] > a[i])
            sorted = false;
    return sorted;
}
```

Số phép so sánh là n-1

Phân tích thuật toán

- Các loại thời gian tính
 - Thời gian tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước n . Thời gian như vậy sẽ được gọi là **thời gian tính tốt nhất** của thuật toán với đầu vào kích thước n
 - Thời gian nhiều nhất cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước n . Thời gian như vậy sẽ được gọi là **thời gian tính tồi nhất** của thuật toán với đầu vào kích thước n
 - Thời gian trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào kích thước n . Thời gian như vậy sẽ được gọi là **thời gian tính trung bình** của thuật toán

Phân tích thuật toán

- Ký hiệu tiệm cận
 - Các ký hiệu Θ , O , Ω
 - Được xác định đối với các hàm nhận giá trị nguyên không âm
 - Dùng để so sánh tốc độ tăng của các hàm
 - Được sử dụng để mô tả thời gian tính của thuật toán. Thay vì nói chính xác, ta có thể nói thời gian tính là, chẳng hạn, $\Theta(n^2)$

Phân tích thuật toán

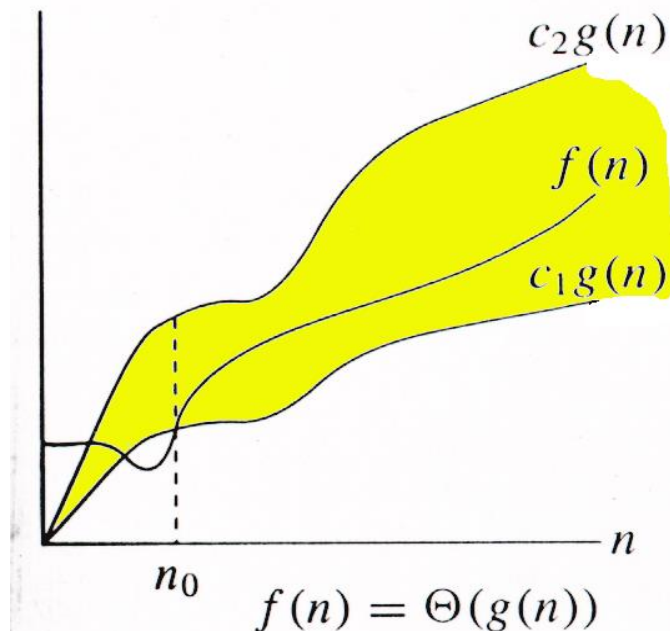
- Ký hiệu Θ

Đối với hàm $g(n)$ cho trước, ta ký hiệu $\Theta(g(n))$ là tập các hàm

$\Theta(g(n)) = \{f(n) \mid \text{tồn tại các hằng số } c_1, c_2 \text{ và } n_0 \text{ sao cho}$

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \text{ với mọi } n \geq n_0 \}$$

Ta nói rằng $g(n)$ là **đánh giá tiệm cận đúng** cho $f(n)$



Phân tích thuật toán

- Ví dụ

- $10n^2 - 3n = \Theta(n^2)$

- Với giá trị nào của các hằng số n_0 , c_1 , và c_2 thì bất đẳng thức sau đây là đúng với $n \geq n_0$:

$$c_1 n^2 \leq 10n^2 - 3n \leq c_2 n^2$$

- Ta có thể lấy c_1 bé hơn hệ số của số hạng với số mũ cao nhất, còn c_2 lấy lớn hơn hệ số này, chẳng hạn:

$$c_1 = 9 < 10 < c_2 = 11, n_0 = 10.$$

- *Tổng quát, để so sánh tốc độ tăng của các đa thức, cần nhìn vào số hạng với số mũ cao nhất*

Phân tích thuật toán

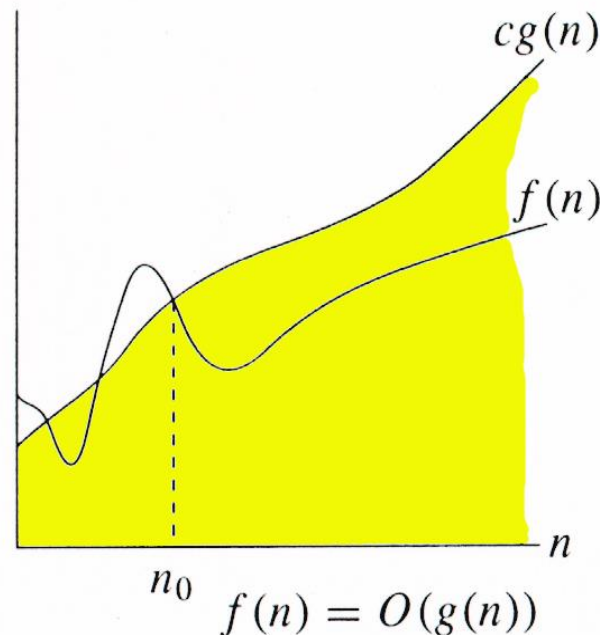
- **Kí hiệu O lớn**

Đối với hàm $g(n)$ cho trước, ta ký hiệu $O(g(n))$ là tập các hàm

$O(g(n)) = \{f(n) \mid \text{tồn tại các hằng số dương } c \text{ và } n_0 \text{ sao cho:}$

$$f(n) \leq c \times g(n) \text{ với mọi } n \geq n_0\}$$

Ta nói $g(n)$ là *cận trên tiệm cận* của $f(n)$



Phân tích thuật toán

- Ví dụ kí hiệu O lớn

- **Chứng minh:** $f(n) = n^2 + 2n + 1$ là $O(n^2)$

- Cần chỉ ra: $n^2 + 2n + 1 \leq c \cdot n^2$

- với c là hằng số nào đó và khi $n \geq n_0$ nào đó

- Ta có:

$$2n^2 \geq 2n \text{ khi } n \geq 1$$

$$\text{và } n^2 \geq 1 \text{ khi } n \geq 1$$

- Vì vậy

$$n^2 + 2n + 1 \leq 4 \cdot n^2 \text{ với mọi } n \geq 1$$

- Như vậy hằng số $c = 4$, và $n_0 = 1$

Phân tích thuật toán

- Ví dụ kí hiệu O lớn

- Ta thấy rằng: Nếu $f(n)$ là $O(n^2)$ thì nó cũng là $O(n^k)$ với $k > 2$.
- **Chứng minh:** $f(n) = n^2 + 2n + 1 \notin O(n)$.
- Phản chứng. Giả sử trái lại, khi đó phải tìm được hằng số c và số n_0 để cho:

$$n^2 + 2n + 1 \leq c \cdot n \text{ khi } n \geq n_0$$

- Suy ra

$$n^2 < n^2 + 2n + 1 \leq c \cdot n \text{ với mọi } n \geq n_0$$

- Từ đó ta thu được:

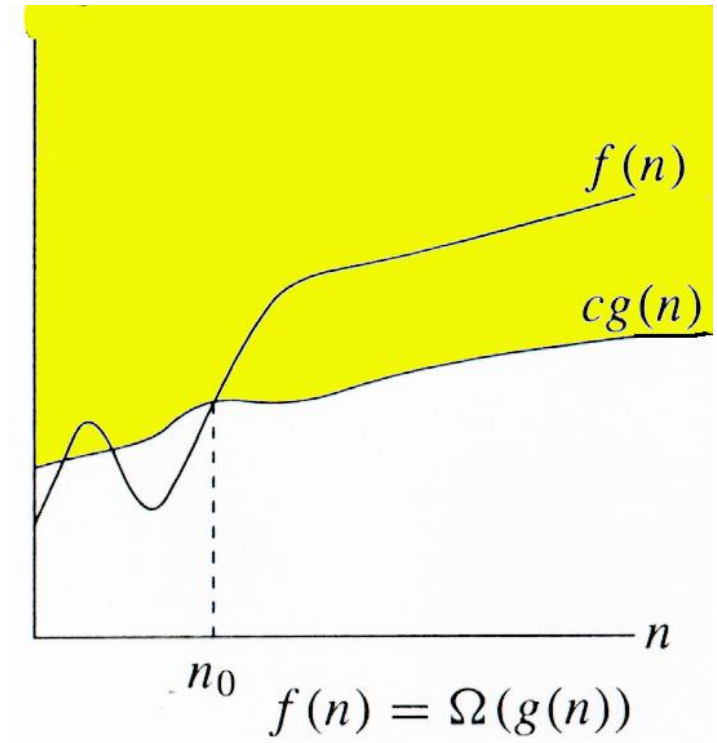
$$n < c \text{ với mọi } n \geq n_0 \quad ?!$$

Phân tích thuật toán

- Kí hiệu Ω

Đối với hàm $g(n)$ cho trước, ta ký hiệu $\Omega(g(n))$ là tập các hàm:

$\Omega(g(n)) = \{f(n) \mid \text{tồn tại các hằng số dương } c \text{ và } n_0 \text{ sao cho}$
 $cg(n) \leq f(n) \text{ với mọi } n \geq n_0\}$



Ta nói $g(n)$ là **cận dưới tiệm cận của $f(n)$**

Phân tích thuật toán

- Ví dụ kí hiệu Ω

- **Chứng minh:** $f(n) = n^2 - 2000n$ là $\Omega(n^2)$

- Cần chỉ ra: $n^2 - 2000n \geq c \cdot n^2$

- với c là hằng số nào đó và khi $n \geq n_0$ nào đó

- Ta có:

$$n^2 - 2000n \geq 0.5 \cdot n^2 \text{ với mọi } n \geq 10000$$

$$(\text{vì } n^2 - 2000n - 0.5 \cdot n^2 = 0.5 \cdot n^2 - 2000n$$

$$= n(0.5 \cdot n - 2000) \geq 0$$

$$\text{khi } n \geq 10000)$$

- Như vậy hằng số $c = 1$, và $n_0 = 10000$

Phân tích thuật toán

- Liên hệ giữa Θ , Ω , O

- Đối với hai hàm bất kỳ $g(n)$ và $f(n)$,

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

khi và chỉ khi

$$f(n) = O(g(n)) \text{ và } f(n) = \Omega(g(n)).$$

tức là

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

Phân tích thuật toán

- Tính chất bắc cầu
 - Nếu $f(n) = O(g(n))$ và $g(n) = O(h(n))$
 $\rightarrow f(n) = O(h(n))$
 - Nếu $f(n) = \Omega(g(n))$ và $g(n) = \Omega(h(n))$
 $\rightarrow f(n) = \Omega(h(n))$
 - Nếu $f(n) = \Theta(g(n))$ và $g(n) = \Theta(h(n))$
 $\rightarrow f(n) = \Theta(h(n))$

Phân tích thuật toán

- **Chú ý**

- Giá trị của n_0 và c **không phải là duy nhất** trong chứng minh công thức tiệm cận
- Chứng minh rằng $100n + 5 = O(n^2)$
 - $100n + 5 \leq 100n + n = 101n \leq 101n^2$ với mọi $n \geq 5$
 $n_0 = 5$ và $c = 101$ là các hằng số cần tìm
 - $100n + 5 \leq 100n + 5n = 105n \leq 105n^2$ với mọi $n \geq 1$
 $n_0 = 1$ và $c = 105$ cũng là các hằng số cần tìm
- Chỉ cần tìm các hằng c và n_0 **nào đó** thoả mãn bất đẳng thức trong định nghĩa công thức tiệm cận

Phân tích thuật toán

- Cách nói về thời gian tính

- Nói “Thời gian tính là $O(f(n))$ ” hiểu là: Đánh giá trong tình huống tồi nhất (worst case) là $O(f(n))$. Thường nói: “Đánh giá thời gian tính trong tình huống tồi nhất là $O(f(n))$ ”
 - Nghĩa là thời gian tính trong tình huống tồi nhất được xác định bởi một hàm nào đó $g(n) \in O(f(n))$
- “Thời gian tính là $\Omega(f(n))$ ” hiểu là: Đánh giá trong tình huống tốt nhất (best case) là $\Omega(f(n))$. Thường nói: “Đánh giá thời gian tính trong tình huống tốt nhất là $\Omega(f(n))$ ”
 - Nghĩa là thời gian tính trong tình huống tốt nhất được xác định bởi một hàm nào đó $g(n) \in \Omega(f(n))$

Phân tích thuật toán

- Tên gọi của một số tốc độ tăng

Hàm	Tên gọi
c	Hằng
$\log n$	Logarit
$\log^2 n$	Logarit bình phương
n	tuyến tính
$n \log n$	$n \log n$
n^2	bình phương
n^3	bậc 3
2^n	hàm mũ
n^k (k là hằng số dương)	đa thức

Phân tích thuật toán

- Hàm nào tăng chậm hơn?
 - $f(n) = n \log n$ và $g(n) = n^{1,5}$
 - Lời giải:
 - Chú ý rằng $g(n) = n^{1,5} = n * n^{0,5}$
 - Vì vậy, chỉ cần so sánh $\log n$ và $n^{0,5}$
 - Tương đương với so sánh $\log^2 n$ và n
 - Hàm $\log^2 n$ tăng chậm hơn n
 - Suy ra $f(n)$ tăng chậm hơn $g(n)$

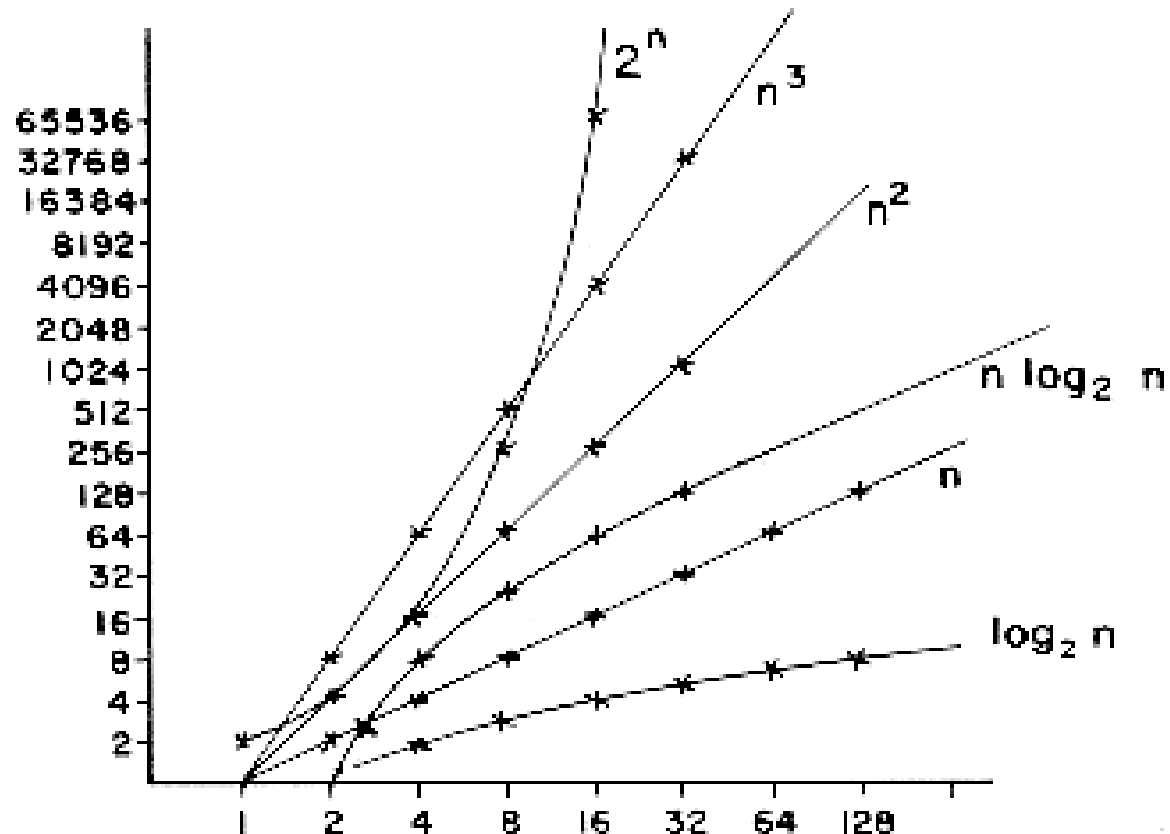
Phân tích thuật toán

- Hàm nào tăng chậm hơn?
 - Xét một chiếc máy tính thực hiện được 1.000.000 phép tính cơ bản trong một giây
 - Khi thời gian chạy vượt quá 10^{25} năm, ta viết "very long"

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Phân tích thuật toán

- Đồ thị của một số hàm cơ bản



Phân tích thuật toán

- Ví dụ 1

```
1 for (i = 0; i < n; i++)
2 {
3     x = a[i] / 2;
4     a[i] = x + 1;
5 }
```

- Có 4 phép tính cơ bản ở các dòng 3 và 4 gồm 2 phép gán, 1 phép chia và 1 phép cộng (*bỏ qua các phép tính cơ bản điều khiển quá trình lặp ở dòng 1*)
- Cả 4 phép tính đó được lặp lại n lần
- Thời gian chạy: $T(n) = 4n = O(n)$

Quy tắc bỏ hằng số: Nếu đoạn chương trình P có thời gian thực hiện $T(n) = O(c1.f(n))$ với $c1$ là một hằng số dương thì có thể coi đoạn chương trình đó có độ phức tạp tính toán là $O(f(n))$

Phân tích thuật toán

- Ví dụ 2

Xét thuật toán tìm kiếm tuần tự để giải bài toán

Đầu vào: n và dãy số s_0, s_1, \dots, s_{n-1} .

Đầu ra: Vị trí phần tử có giá trị key hoặc là n nếu không tìm thấy

```
void Linear_Search(s, n, key) {  
    i = -1;  
    do {  
        i = i + 1;  
    } while (i < n) || (key == s[i]);  
    return i;  
}
```

Phân tích thuật toán

• Ví dụ 2

- Cần đánh giá thời gian tính tốt nhất, tồi nhất, trung bình của thuật toán với độ dài đầu vào là n .
- Thời gian tính của thuật toán có thể được đánh giá bởi số lần thực hiện câu lệnh $i = i + 1$ trong vòng lặp **do** (*thực tế là câu lệnh này có hai phép tính cơ bản là **phép gán** và **phép cộng**. Ta cũng bỏ qua hai phép so sánh của vòng lặp **do***)
- Nếu $s_0 = \text{key}$ thì câu lệnh $i = i + 1$ trong thân vòng lặp **do** thực hiện 1 lần. Do đó thời gian tính tốt nhất của thuật toán là $\Omega(1)$
- Nếu key không có mặt trong dãy đã cho, thì câu lệnh $i = i + 1$ thực hiện n lần. Vì thế thời gian tính tồi nhất của thuật toán là $O(n)$

Phân tích thuật toán

• Ví dụ 2

- Cuối cùng, ta tính thời gian tính trung bình của thuật toán
 - Nếu *key* tìm thấy ở vị trí thứ *i* của dãy (*key* = *s_i*) thì câu lệnh *i* = *i*+1 phải thực hiện *i* lần (*i* = 1, 2, ..., *n*),
 - Nếu *key* không có mặt trong dãy đã cho thì câu lệnh *i* = *i*+1 phải thực hiện *n* lần.
- Từ đó suy ra số lần trung bình phải thực hiện câu lệnh *i* = *i*+1 là

$$[(1 + 2 + \dots + n) + n] / (n+1) = [n(n+1)/2 + n] / (n+1)$$

- Ta có

$$n/4 \leq [n(n+1)/2 + n] / (n+1) \leq n \text{ với mọi } n \geq 1$$

- Vậy thời gian tính trung bình của thuật toán là $\Theta(n)$

Phân tích thuật toán

- Ví dụ 3: vòng lặp có câu lệnh break

```
1 for (i = 0; i < n; i++)  
2 {  
3     x = a[i] / 2;  
4     a[i] = x + 1;  
5     if (a[i] > 10) break;  
6 }
```

- Có 5 phép tính cơ bản ở các dòng 3, 4, 5 gồm 2 phép gán, 1 phép chia, 1 phép cộng và 1 phép so sánh (bỏ qua vòng for)
- Không thể đếm chính xác số lần thực hiện 5 thao tác đó vì ta không biết khi nào điều kiện $a[i] > 10$ xảy ra
- Trường hợp tồi nhất, điều kiện $a[i] > 10$ xảy ra ở bước lặp cuối cùng hoặc không xảy ra, cả 5 thao tác cơ bản được lặp lại n lần
- Thời gian chạy trong trường hợp tồi nhất: $T(n) = 5n = O(n)$

Phân tích thuật toán

- Ví dụ 4: Các vòng lặp tuần tự

```
1  for (i = 0; i < n; i++)
2  {
3      // 3 phép tính cơ bản trong vòng lặp
4  }
5  for (i = 0; i < n; i++)
6  {
7      // 5 phép tính cơ bản trong vòng lặp
8  }
```

- Chỉ cần cộng thời gian chạy của các vòng lặp
- Thời gian chạy tổng thể: $T(n) = 3n + 5n = 8n = O(n)$

Quy tắc cộng – lấy max: Nếu $T_1(n)$ và $T_2(n)$ là thời gian thực hiện của hai đoạn chương trình P1 và P2; và $T_1(n)=O(f(n))$, $T_2(n)=O(g(n))$ thì thời gian thực hiện của đoạn hai chương trình nối tiếp nhau đó là $T(n)=O(\max(f(n),g(n)))$

Phân tích thuật toán

- Ví dụ 5: Các vòng lặp lồng nhau

```
1  for (i = 0; i < n; i++) {  
2      // 3 phép tính cơ bản ở đây  
3      for (j=0; j < n; j++) {  
4          // 3 phép tính cơ bản trong vòng lặp  
5      }  
6  }
```

- Vòng lặp bên trong thực hiện $3n$ phép tính cơ bản
- Mỗi bước lặp của vòng lặp bên ngoài thực hiện $3n$ phép tính cơ bản
- Thời gian chạy tổng thể: $T(n) = 3n \cdot 3n = 9n^2 = O(n^2)$

Quy tắc nhân: Nếu $T1(n)$ và $T2(n)$ là thời gian thực hiện của hai đoạn chương trình P1 và P2 và $T1(n) = O(f(n))$, $T2(n) = O(g(n))$ thì thời gian thực hiện của đoạn hai đoạn chương trình lồng nhau đó là $T(n) = O(f(n) \times g(n))$

Phân tích thuật toán

- Bài tập

1. Sắp xếp danh sách các hàm sau đây theo thứ tự tăng dần của tốc độ tăng trưởng

$$f_1(n) = n^{2,5}$$

$$f_2(n) = \sqrt{2n}$$

$$f_3(n) = n + 10$$

$$f_4(n) = 10^n$$

$$f_5(n) = 100^n$$

$$f_6(n) = n^2 \log n$$

Phân tích thuật toán

- Bài tập

2. Phân tích thời gian chạy của thuật toán tìm phần tử lớn nhất (dùng ký hiệu O)

```
max = a[0];  
for (i = 1; i < n; i++)  
    if (a[i] > max)  
        max = a[i];
```

Phân tích thuật toán

- Bài tập

3. Phân tích thời gian chạy của thuật toán hợp nhất hai danh sách đã sắp xếp $A = a_1, a_2, \dots, a_n$ và $B = b_1, b_2, \dots, b_n$ thành một danh sách tổng thể C cũng được sắp xếp

$i = 0; j = 0;$

while (cả A và B không rỗng) {

 if ($a_i \leq b_j$)

 Thêm a_i vào cuối C và tăng i một đơn vị;

 else

 Thêm b_j vào cuối C và tăng j một đơn vị

}

Thêm phần còn lại của danh sách không rỗng vào cuối C

Phân tích thuật toán

- Bài tập

4. Hãy phân tích thời gian chạy của thuật toán sau

```
for (i = 0; i < n; i++) {  
    for (j = i + 1; j < n; j++) {  
        Cộng các phần tử A[i] và A[j];  
        Lưu trữ kết quả vào B[i,j];  
    }  
}
```

Phân tích thuật toán

- Bài tập

5. Phân tích thời gian chạy của các đoạn chương trình C++ sau đây

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

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

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

Đánh giá độ phức tạp thuật toán

- **Phân tích các chương trình đệ quy**
 - Với các chương trình đệ quy có gọi các chương trình đệ quy con, ta không thể áp dụng cách tính như trên được vì một chương trình đệ quy sẽ gọi chính bản thân nó
 - Cần lập thành các phương trình đệ quy, sau đó giải phương trình đệ quy, nghiệm của phương trình đệ quy chính là thời gian thực hiện của chương trình đệ quy đó

Đánh giá độ phức tạp thuật toán

- **Thành lập phương trình đệ quy**
 - Phương trình đệ quy là một phương trình biểu diễn mối liên hệ giữa $T(n)$ và $T(k)$
 - $T(n)$ là thời gian thực hiện chương trình với kích thước dữ liệu đầu vào là n
 - $T(k)$ là thời gian thực hiện chương trình với kích thước dữ liệu đầu vào là k với $k < n$
 - Để thành lập phương trình đệ quy ta phải căn cứ vào chương trình đệ quy

Đánh giá độ phức tạp thuật toán

- Thành lập phương trình đệ quy

- Xét hàm tính giai thừa bằng thuật toán đệ quy

```
int Giai_thua(int n)
{
    if (n == 0) return 1;
    else return n * Giai_thua(n-1);
}
```

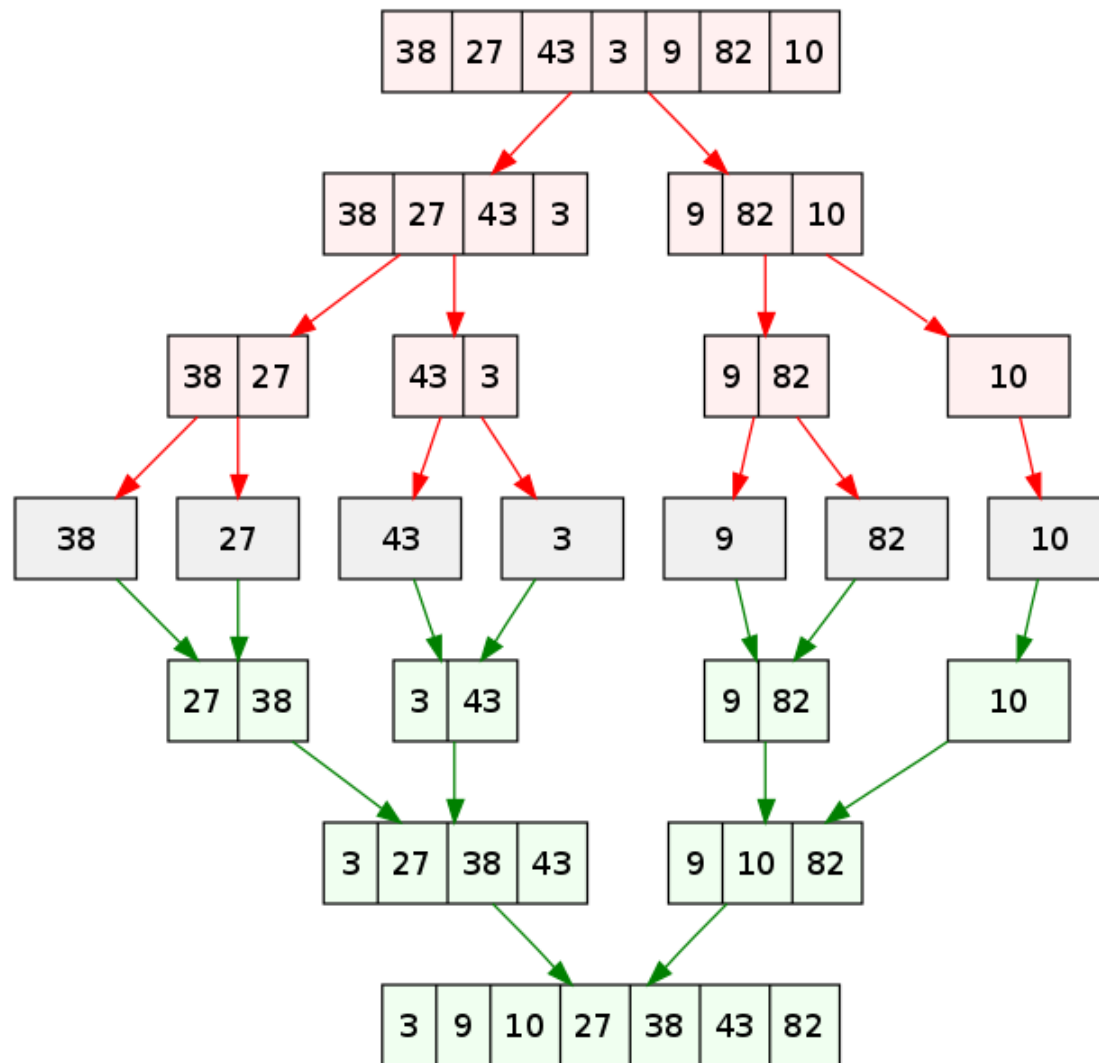
- $T(n)$ thời gian thực hiện tính n giai thừa
- $T(n-1)$ thời gian thực hiện tính $n - 1$ giai thừa
- $T(0) = C_1$. Với $n > 0$, gọi đệ quy tốn $T(n-1)$
- Thao tác nhân n với kết quả đệ quy là C_2
- Phương trình đệ quy:
$$T(n) = \begin{cases} C_1 & \text{nếu } n = 0 \\ T(n-1) + C_2 & \text{nếu } n > 0 \end{cases}$$

Đánh giá độ phức tạp thuật toán

- **Thành lập phương trình đệ quy**
 - Xét thuật toán sắp xếp trộn (merge sort)
 - **Chia:** Chia mảng $A[1, 2, \dots, n]$ thành hai phần có kích thước xấp xỉ nhau
 - **Trị:** Gọi đệ quy sắp xếp trên mỗi phần
 - **Trộn:** trộn hai mảng con đã được sắp xếp với nhau thành mảng được sắp xếp cho bước đệ quy hiện tại

Đánh giá độ phức tạp thuật toán

- Thành lập phương trình đệ quy
 - Xét thuật toán sắp xếp trộn (merge sort)



Đánh giá độ phức tạp thuật toán

- **Thành lập phương trình đệ quy**
 - Xét thuật toán sắp xếp trộn (merge sort)
 - $T(n)$ thời gian thực hiện sắp xếp trộn n phần tử
 - $T(n/2)$ thời gian thực hiện sắp xếp trộn $n/2$ phần tử
 - C_1 là thời gian sắp xếp với danh sách có độ dài 1
 - Với $n > 1$, chia danh sách gồm hai nửa và trộn nên ta có tổng thời gian là C_2n
 - Phương trình đệ quy:
$$T(n) = \begin{cases} C_1 & \text{nếu } n = 1 \\ 2T(n/2) + C_2n & \text{nếu } n > 1 \end{cases}$$

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy
 - Có ba phương pháp giải phương trình đệ quy
 - Phương pháp truy hồi
 - Phương pháp đoán nghiệm
 - Lời giải tổng quát của một lớp phương trình đệ quy

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy
 - Phương pháp truy hồi
 - Dùng đệ quy để thay thế bất kỳ $T(m)$ với $m < n$ vào phía phải của phương trình cho đến khi tất cả $T(m)$ với $m > 1$ được thay thế bởi biểu thức của các $T(1)$
 - Vì $T(1)$ luôn là hằng nên chúng ta có công thức của $T(n)$ chứa các số hạng chỉ liên quan đến n và các hằng số

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy

- Phương pháp truy hồi

- Giải phương trình đệ quy

$$T(n) = \begin{cases} C_1 & \text{nếu } n = 1 \\ 2T(n/2) + C_2n & \text{nếu } n > 1 \end{cases}$$

- Ta có:

$$T(n) = 2T(n/2) + C_2n$$

$$T(n) = 2[2T(n/4) + C_2n/2] + C_2n = 4T(n/4) + 2C_2n$$

$$T(n) = 4[2T(n/8) + C_2n/4] + 2C_2n = 8T(n/8) + 3C_2n$$

$$T(n) = 2^i T(n/2^i) + iC_2n$$

- Giả sử $n = 2^k$, quá trình suy rộng kết thúc khi $i = k$, khi đó: $T(n) = 2^k T(1) + kC_2n$. Vì $2^k = n$ nên $k = \log n$ và với $T(1) = C_1$ nên $T(n) = C_1n + C_2n \log n$ hay $T(n) = O(n \log n)$

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy
 - Đoán nghiệm
 - Đoán một nghiệm $f(n)$ và dùng chứng minh quy nạp để chứng tỏ rằng $T(n) \leq f(n)$ với mọi n
 - $f(n)$ là một trong các hàm quen thuộc như $\log n$, n , $n \log n$, n^2 , n^3 , $2n$, $n!$, n^n
 - Đôi khi ta chỉ đoán dạng của $f(n)$ trong đó có một vài tham số chưa xác định (ví dụ, $f(n) = an^2$ với a chưa xác định) và trong quá trình chứng minh quy nạp ta suy diễn ra giá trị thích hợp của các tham số

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy
 - Lời giải tổng quát cho một lớp phương trình đệ quy
 - Để giải bài toán kích thước n , ta chia thành a bài toán con và mỗi bài toán con có kích thước n/b . Giải các bài toán con này và tổng hợp kết quả để được kết quả của bài toán ban đầu
 - Đối với các bài toán con, ta cũng làm tương tự
→ một chương trình đệ quy

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy
 - Lời giải tổng quát cho một lớp phương trình đệ quy
 - Giả sử mỗi bài toán con kích thước 1 lấy một đơn vị thời gian, $d(n)$ là thời gian để chia bài toán kích thước n thành các bài toán con kích thước n/b và tổng hợp kết quả từ các bài toán con
 - Gọi $T(n)$ là thời gian để giải bài toán kích thước n thì ta có phương trình đệ quy:

$$\begin{cases} T(1) = 1 \\ T(n) = aT\left(\frac{n}{b}\right) + d(n) \end{cases} \quad (I.1)$$

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy
 - Lời giải tổng quát cho một lớp phương trình đệ quy
 - Sử dụng phương pháp truy hồi để giải phương trình

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

$$T(n) = a[aT(n/b^2) + d(n/b)] + d(n) = a^2T(n/b^2) + ad(n/b) + d(n)$$

$$T(n) = a^2[aT(n/b^3) + d(n/b^2)] + ad(n/b) + d(n)$$

$$= a^3T(n/b^3) + a^2d(n/b^2) + ad(n/b) + d(n)$$

$$= \dots$$

$$= a^iT(n/b^i) + \sum_{j=0}^{i-1} a^j d(n/b^j)$$

- Giả sử $n = b^k$ ta được $T(n/b^k) = T(1) = 1$. Thay vào trên với $i = k$ ta có: $T(n) = a^k + \sum_{j=0}^{k-1} a^j d(b^{k-j})$ (I.2)

Đánh giá độ phức tạp thuật toán

• Giải phương trình đệ quy

- Lời giải tổng quát cho một lớp phương trình đệ quy
 - Sử dụng phương pháp truy hồi để giải phương trình
 - Trong phương trình đệ quy (I.1), hàm thời gian $d(n)$ được gọi là hàm tiến triển
 - Trong công thức (I.2), $a^k = n^{\log_b a}$ được gọi là **nghiệm thuần nhất** → là nghiệm chính xác khi $d(n) = 0$ với mọi n
→ biểu diễn thời gian để giải tất cả các bài toán con
 - Trong công thức (I.2), $\sum_{j=0}^{k-1} a^j d(b^{k-j})$ được gọi là **nghiệm riêng** → biểu diễn thời gian phải trả để tạo các bài toán con và tổng hợp kết quả của chúng
→ phụ thuộc vào hàm tiến triển, số lượng và kích thước bài toán con

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy
 - Lời giải tổng quát cho một lớp phương trình đệ quy
 - Sử dụng phương pháp truy hồi để giải phương trình
 - Khi tìm nghiệm của phương trình (I.1) ta phải tìm nghiệm riêng và so sánh với nghiệm thuần nhất.
Nghiệm của phương trình (I.1) sẽ là nghiệm lớn hơn
 - Việc xác định nghiệm riêng không dễ chút nào. Tuy nhiên, chúng ta tìm được một lớp các hàm tiến triển có thể dễ dàng xác định nghiệm riêng

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy

- Hàm nhân

- Một hàm được gọi là hàm nhân nếu $f(m.n) = f(m).f(n)$ với m, n là các số nguyên dương
 - Ví dụ: hàm $f(n) = n^k$ là một hàm nhân vì:
$$f(m.n) = (m.n)^k = m^k.n^k = f(m).f(n)$$
 - Nếu $d(n)$ trong (I.1) là một hàm nhân thì ta có:
$$d(b^{k-j}) = (d(b))^{k-j}$$
 và nghiệm riêng của (I.2) là:

$$(I.3) \quad \sum_{j=0}^{k-1} a^j d(b^{k-j}) = d(b)^k \sum_{j=0}^{k-1} \left[\frac{a}{d(b)} \right]^j = d(b)^k \frac{\left[\frac{a}{d(b)} \right]^k - 1}{\frac{a}{d(b)} - 1} = \frac{a^k - d(b)^k}{\frac{a}{d(b)} - 1}$$

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy

- Xét ba trường hợp sau:

1) Nếu $a > d(b)$ thì nghiệm riêng là: $O(a^k) = O(n^{\log_b a})$

Nghiem riêng = nghiệm thuần nhất: $T(n) = O(n^{\log_b a})$

→ thời gian thực hiện chỉ phụ thuộc vào a và b mà không phụ thuộc vào hàm tiến triển $d(n)$. Do đó, để cải thiện thuật toán ta cần giảm a và tăng b

2) $a < d(b)$ thì $O(d(b)^k) = O(n^{\log_b d(b)})$. Nghiệm riêng lớn hơn nghiệm thuần nhất, khi đó: $T(n) = O(n^{\log_b d(b)})$

Trường hợp đặc biệt khi $d(n) = n^\alpha$. Khi đó $d(b) = b^\alpha$ và $\log_b(b^\alpha) = \alpha \rightarrow$ nghiệm riêng là $O(n^\alpha)$ và $T(n) = O(n^\alpha)$

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy

- Xét ba trường hợp sau:

3) Nếu $a = d(b)$ thì công thức (I.3) không xác định. Tính trực tiếp nghiệm riêng:

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

$$= d(b)^k k = n^{\log_b d(b)} \log_b n$$

Vì $a = d(b)$ nên nghiệm riêng là: $n^{\log_b a} \log_b n$ và nghiệm này lớn gấp $\log_b n$ nghiệm thuần nhất nên:

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

Đặc biệt khi $d(n) = n^\alpha \rightarrow T(n) = O(n^\alpha \log n)$

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy
 - Giải một phương trình đệ quy cụ thể
 - Xem dạng có thuộc dạng phương trình tổng quát không
 - Nếu có thì xem hàm tiến triển có phải là hàm nhân không
 - Nếu có thì ta xác định a , $d(b)$ và sự so sánh giữa a và $d(b)$ để vận dụng một trong ba trường hợp nêu trên

Đánh giá độ phức tạp thuật toán

- Giải phương trình đệ quy
 - Ví dụ: giải các phương trình đệ quy với $T(1) = 1$
 - 1) $T(n) = 4T(n/2) + n$
 - 2) $T(n) = 4T(n/2) + n^2$
 - 3) $T(n) = 4T(n/2) + n^3$
 - Với mỗi trường hợp, $a = 4$, $b = 2$ và nghiệm thuần nhất là $n^{\log_2 4} = n^2$
 - Trong 1), $d(n) = n \rightarrow d(b) = 2$. Vì $a = 4 > d(b) = 2$ nên nghiệm riêng cũng là $n^2 \rightarrow T(n) = O(n^2)$
 - Trong 2), $d(b) = 4 = a \rightarrow T(n) = O(n^2 \log n)$
 - Trong 3), $d(n) = n^3$, $d(b) = 8$ và $a < d(b) \rightarrow$ nghiệm riêng là $O(n^{\log_b d(b)}) = O(n^{\log_2 8}) = O(n^3) = T(n)$

Đánh giá độ phức tạp thuật toán

- Định lý thợ (master theorem)

- Dùng để giải các phương trình đệ quy có dạng sau:

$T(n) = aT(n/b) + f(n)$, trong đó $a \geq 1$ và $b > 1$

- Có ba trường hợp sau đây:

1) Nếu $f(n) = O(n^\alpha)$ và $\alpha < \log_b a$ thì $T(n) = O(n^{\log_b a})$

2) Nếu $f(n) = O(n^\alpha)$ và $\alpha = \log_b a$ thì $T(n) = O(n^\alpha \log n)$

3) Nếu $f(n) = O(n^\alpha)$ và $\alpha > \log_b a$ thì $T(n) = O(f(n))$

- Ví dụ:

- Với **merge sort**: $T(n) = 2T(n/2) + O(n) \rightarrow$ rơi vào trường hợp 2 khi $\alpha = \log_b a = 1 \rightarrow T(n) = O(n \log n)$

- Với **binary search**: $T(n) = T(n/2) + O(1) \rightarrow$ rơi vào trường hợp 2 khi $\alpha = \log_b a = 0 \rightarrow T(n) = O(\log n)$

Bài tập

- Giải các phương trình đệ quy sau với $T(1) = 1$ và
 - $T(n) = 3T(n/2) + n$
 - $T(n) = 3T(n/2) + n^2$
 - $T(n) = 8T(n/2) + n^3$
 - $T(n) = 4T(n/3) + n$
 - $T(n) = 4T(n/3) + n^2$
 - $T(n) = 9T(n/3) + n^2$
 - $T(n) = 2T(n/2) + n$
 - $T(n) = 2T(n/2) + n^2$
 - $T(n) = 2T(n/2) + \log n$

TỔNG KẾT