CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Chương 1. GIỚI THIỆU

1.1. Thuật toán và giải thuật

1.2. Cấu trúc dữ liệu

1.3. Độ phức tạp thuật toán

1.4. Một số giải thuật cơ bản

1.5. CASE STUDY

Chương 2. Một số giải thuật quan trọng

2.1. Thuật toán vét cạn

2.2. Thuật toán sinh (Generation Algorithm)

2.3. Thuật toán đệ qui (Recursive Algorithm

2.4. Thuật toán quay lui (Back Track Alogorithm)

2.5. Thuật toán tham lam (Greedy Algorithm)

2.6. Thuật toán chia để trị (Divide and Conquer Algorithm)

2.7. Thuật toán qui hoạch động (Dynamic Algorithm)

2.8. Thuật toán nhánh cận (Branch an Bound)

2.9. Thuật toán tìm kiếm mẫu (Pattern Searching)

2.10. CASE STUDY

Chương 3. Stack – Queue – Link List

- 3.1. Một số thuật toán dựa vào ngăn xếp
- 3.2. Một số Thuật toán dựa vào hàng đợi
- 3.3. Một số Thuật toán dựa trên danh sách liên kết
- 3.4. Khử đệ qui dựa vào ngăn xếp
- 3.5. Ứng dụng của ngăn xếp
- 3.6. Ứng dụng của hàng đợi
- 3.7. Ứng dụng của danh sách liên kết
- 3.8. CASE STUDY

Chương 4. Cây nhị phân

- 4.1. Một số thuật thuật ngữ
- 4.2. Các thao tác trên cây nhị phân
- 4.3. Cây nhị phân tìm kiếm
- 4.4. Cây nhị phân cân bằng
- 4.5. B-Tree
- 4.6. Cây đỏ đen
- 4.7. Cây tổng quát
- 4.8. CASE STUDY

Chương 5. Các thuật toán trên đồ thị

5.1. Một số thuật toán trên đồ thị vô hướng

5.2. Một số thuật toán trên đồ thị có hướng

5.3. Một số thuật toán trên đồ thị Euler

5.4. Một số thuật toán trên đồ thị Hamilton

5.5. Một số thuật toán trên đồ thị đầy đủ

5.6. Một số thuật toán trên đồ thị hai phía

5.7. Luồng cực đại trên mạng

5.8. CASE STUDY

Chương 6. Sắp xếp và tìm kiếm

- 6.1. Sắp xếp đơn giản
- 6.2. Sắp xếp nhanh (Quick Sort)
- 6.3. Sắp xếp kiểu hòa nhập (Merge Sort)
- 6.4. Sắp xếp kiểu vun đống (Heap Sort)
- 6.5. Tìm kiếm tuyến tính
- 6.6. Tìm kiếm nhị phân
- 6.7. Tìm kiếm theo cơ số
- 6.8. Tìm kiếm trên cây nhị phân
- 6.9. CASE STUDY

TÀI LIỆU THAM KHẢO:

[1] Robert Sedgewick, “Algorihms”, McGraw Hill Company, 2006, Vol 1, 2, 3, 4.

[2] N. Knuth, “The Art of Programming”, McGraw Hill Company, 2006, Vol 1, 2, 3, 4.

[3] Robert Sedgewick, “Cẩm nang thuật toán”, Nhà xuất bản khoa học kỹ thuật Hà Nội, 2004, Vol 1, 2, 3, 4.

[4] Lê Minh Hoàng, “Bài giảng các chuyên đề”, Nhà xuất bản khoa học kỹ thuật Hà Nội, 2008.

YÊU CẦU:

1. Hiểu phương pháp biểu diễn thuật toán.
2. Đánh giá độ phức tạp thuật toán.
3. Cài đặt thuật toán bằng ngôn ngữ C++.
4. Hoàn thành CASE STUDY theo mỗi chương.

PHƯƠNG PHÁP ĐÁNH GIÁ:

- LÊN LỚP ĐẦY ĐỦ : 10%
- THỰC HÀNH ONLINE : 10%
- THAM GIA SUBCONTEST : 10%
- THI CUỐI KỲ : 10%

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

NỘI DUNG:

Chương 1. MỞ ĐẦU

- 1.1. Qui trình giải quyết vấn đề bằng máy tính
- 1.2. Thuật toán và giải thuật
- 1.3. Cấu trúc dữ liệu
- 1.4. Độ phức tạp thuật toán
- 1.5. Một số thuật toán cơ bản
- 1.6. CASE STUDY

1.1. Qui trình giải quyết vấn đề bằng máy tính

•**Xác định yêu cầu bài toán**: Xem xét bài toán cần xử lý vấn đề gì? Giả thiết nào đã được biết trước và lời giải cần đạt những yêu cầu gì? Ví dụ thời gian, hay không gian nhớ.

•**Tìm cấu trúc dữ liệu biểu diễn bài toán**: Cấu trúc dữ liệu phải biểu diễn đầy đủ các đối tượng thông tin vào của bài toán. Các thao tác trên cấu trúc dữ liệu phải phù hợp với những thao tác của thuật toán được lựa chọn. Cấu trúc dữ liệu phải cài đặt được bằng ngôn ngữ lập trình cụ thể đáp ứng yêu cầu bài toán.

•**Tìm thuật toán**: Tìm thuật toán hiệu quả ứng với các cấu trúc dữ liệu đã được lựa chọn:

• **Cài đặt thuật toán:** Cài đặt thuật toán trên cấu trúc dữ liệu đã lựa chọn.

• **Kiểm thử chương trình** : Chương trình có thỏa mãn tất cả những yêu cầu đặt ra của bài toán hay không?

• **Tối ưu chương trìn**h: Cải tiến để chương trình tốt hơn.

1.2. Thuật toán và giải thuật

1.2.1. Định nghĩa thuật toán (Algorithm): Thuật toán F giải bài toán P là dãy các thao tác sơ cấp F_1, F_2,..,F_N trên tập dữ kiện đầu vào (Input) để đưa ra được kết quả ra (Output).

$$F_1F_2..F_N(\text{Input}) \rightarrow \text{Ouput}.$$

- $F = F_1F_2..F_N$ được gọi là thuật toán giải bài toán P. Trong đó, mỗi F_i chỉ là các phép tính toán số học hoặc logic.
- Input được gọi là tập dữ kiện đầu vào (dữ liệu đầu vào).
- Output là kết quả nhận được sau khi thực hiện thuật toán F trên tập Input.

Ví dụ. Thuật toán tìm USCLN(a, b).

```
Int     USCLN ( int a,  int  b) {
        while (b!=0 ) {
                x = a % b; a = b; b =x;
        }
        return(a);
}
```

1.2.2. Các đặc trưng của thuật toán:

Một thuật toán cần thỏa mãn các tính chất dưới đây:

- **Tính đơn định**. Ở mỗi bước của thuật toán, các thao tác sơ cấp phải hết sức rõ ràng, không gây nên sự lộn xộn, nhập nhằng, đa nghĩa. Thực hiện đúng các bước của thuật toán trên tập dữ liệu vào, chỉ cho duy nhất một kết quả ra.
- **Tính dừng**. Thuật toán không được rơi vào quá trình vô hạn. Phải dừng lại và cho kết quả sau một số hữu hạn các bước.
- **Tính đúng**. Sau khi thực hiện tất cả các bước của thuật toán theo đúng qui trình đã định, ta phải nhận được kết quả mong muốn với mọi bộ dữ liệu đầu vào. Kết quả đó được kiểm chứng bằng yêu cầu của bài toán.
- **Tính phổ dụng**. Thuật toán phải dễ sửa đổi để thích ứng được với bất kỳ bài toán nào trong lớp các bài toán cùng loại và có thể làm việc trên nhiều loại dữ liệu khác nhau.
- **Tính khả thi**. Thuật toán phải dễ hiểu, dễ cài đặt, thực hiện được trên máy tính với thời gian cho phép.

1.2.3. Khái niệm giải thuật

Giải thuật là khái niệm mở rộng của thuật toán, trong đó tính "*xác định*" được làm mềm đi. Ví dụ dưới đây sẽ minh họa cho điều này.

Ví dụ. Tìm USCLN (a, b).

```
Int         USCLN ( int  a, int  b) {
            if ( a > b ) return ( USCLN( a-b, b));
            else if ( a < b) return (USCLN( a, b-a)) ;
            return(a);
}
```

Giả sử ta tìm USCLN(8, 12) = USCLN(8, 4); //Chưa xác định

= USCLN (4,4); // Chưa xác định

= 4.

Chính vì lý do trên khái niệm giải thuật được thay thế cho khái niệm thuật toán. Tuy vậy, do thói quen người ta vẫn sử dụng đồng nhất các khái niệm này. Từ nay về sau, ta cũng sử dụng chung khái niệm thuật toán và giải thuật là giống nhau và đều có nghĩa: Algorithm, Method, Procedure, Proccess.

1.3. Phương pháp biểu diễn thuật toán:

Thông thường, để biểu diễn một thuật toán ta có thể sử dụng các phương pháp sau:

- **Biểu diễn bằng ngôn ngữ tự nhiên**. Ngôn ngữ tự nhiên là phương tiện giao tiếp giữa con người với con người. Ta có thể sử dụng chính ngôn ngữ này vào việc biểu diễn thuật toán.
- **Ngôn ngữ hình thức**. Ngôn ngữ hình thức là phương tiện giao tiếp giữa con người và hệ thống máy tính. Ví dụ ngôn ngữ sơ đồ khối, ngôn ngữ tựa tự nhiên, ngôn ngữ đặc tả. Đặc điểm chung của các loại ngôn ngữ này là việc sử dụng nó rất gần với ngôn ngữ máy tính.
- **Ngôn ngữ máy tính**. Là phương tiện giao tiếp giữa máy tính và máy tính. Trong trường hợp này ta có thể sử dụng bất kỳ nôn ngữ lập trình nào để mô tả thuật toán.

Ghi chú. Trong các phương pháp biểu diễn thuật toán, phương pháp biểu diễn bằng ngôn ngữ hình thức được sử dụng rộng dãi vì nó gần với ngôn ngữ tự nhiên và gần với ngôn ngữ hình thức.

Ví dụ. Biểu diễn thuật toán tìm USCLN (a, b).

Biểu diễn bằng ngôn ngữ tự nhiên.

Đầu vào (Input). Hai số tự nhiên a, b.

Đầu ra (Output). Số nguyên u lớn nhất để a và b đều chia hết cho u.

Thuật toán (Euclide Algorithm):

Bước 1. Đưa vào hai số tự nhiên a và b.

Bước 2. Nếu b$\neq$ 0 thì chuyển đến bước 3, nếu b=0 thì thực hiện bước 4.

Bước 3. Đặt r = a mod b; a = b; b = r ; Quay quay trở lại bước 2.

Bước 4 (Output). Kết luận u=a là số nguyên cần tìm.

Ví dụ. Biểu diễn thuật toán tìm USCLN (a, b).

Biểu diễn bằng ngôn ngữ hình thức.

Đầu vào (*Input*): a$\in$N, a$\in$N.

Đầu ra (*Output*): u = max { u $\in$N : a mod u =0 and b mod u =0}.

Format : u = Euclide (a, b).

Actions (Euclide Algorithm):

< Nhập a $\in$N và b $\in$N>.

while (b $\neq$ 0) do

r = a mod b;

a = b;

b = r;

endwwhile;

return(a);

Endactions.

Ví dụ. Biểu diễn thuật toán tìm USCLN (a, b).

Biểu diễn bằng ngôn ngữ máy tính (C++).

```
Int     USCLN( int  a, int  b) {
        while ( a != 0 ) {
                r = a % b;
                a = b;
                b = r;
        }
        return(a);
}
```

1.4. Độ phức tạp tính toán

Một bài toán có thể thực hiện bằng nhiều thuật toán khác nhau. Chọn giải thuật nhanh nhất giải bài toán là một nhu cầu của thực tế. Vì vậy ta cần phải có sự ước lượng cụ thể để minh chứng bằng toán học mức độ nhanh chậm của mỗi giải thuật.

1.4.1. Khái niệm độ phức tạp thuật toán

Thời gian thực hiện một giải thuật bằng chương trình máy tính phụ thuộc vào các yếu tố:

- Kích thước dữ liệu vào: Dữ liệu càng lớn thì thời gian xử lý càng chậm.
- Phần cứng máy tính: máy có tốc độ cao thực hiện nhanh hơn trên máy có tốc độ thấp. Tuy vậy, yếu tố này không ảnh hưởng đến quá trình xác định thời gian thực hiện của thuật toán nếu xem xét thời gian thực hiện thuật toán như một hàm của độ dài dữ liệu T(n).

Tổng quát, *cho hai hàm f(x), g(x) xác định trên tập các số nguyên dương hoặc tập các số thực vào tập các số thực. Hàm f(x) được gọi là O(g(x)) nếu tồn tại một hằng số C>0 và n0 sao cho:*

$$|f(x)| \leq C.|g(x)| \text{ với mọi } x \geq n0.$$

Điều này có nghĩa với các giá trị x ≥n0 hàm f(x) bị chặn trên bởi hằng số C nhân với g(x). Nếu f(x) là thời gian thực hiện của một thuật toán thì ta nói giải thuật đó có cấp g(x) hay độ phức tạp thuật toán là O(g(x)).

Ghi chú. Các hằng số C, n0 thỏa mãn điều kiện trên là không duy nhất. Nếu có đồng thời f(x) là O(g(x)) và h(x) thỏa mãn g(x) < h(x) với x>n0 thì ta cũng có f(x) là O(h(n)).

Ví dụ 1. Cho $$f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

Trong đó, ai là các số thực (i =0,1, 2, ..,n). Khi đó $f(x) = O(x^n)$.

Chứng minh. Thực vậy, với mọi x>1:

$$|f(x)| = |a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0|$$

$$\cdots \le |a_n| x^n + |a_{n-1}| x^{n-1} + ... + |a_1| x + |a_0|$$

$$\cdots \le |a_n| x^n + |a_{n-1}| x^n + ... + |a_1| x^n + |a_0| x^n$$

$$\cdots \le x^n (|a_n| + |a_{n-1}| + ... + |a_1| + |a_0|)$$

$$\cdots \le C.x^n = O(x^n).$$

$$C = (|a_n| + |a_{n-1}| + ... + |a_1| + |a_0|)$$

Ví dụ 2. Tìm độ phức tạp thuật toán sắp xếp kiểu Bubble-Sort?

```
Void Bubble-Sort ( int  A[],  int n ) {
        for ( i=1; i<n; i++) {
                for ( j = i+1; j<=n; j++){
                        if (A[i] > A[j]) {
                                t = A[i]; A[i] = A[j]; A[j] = t;
                        }
                }
        }
}
```

Lời giải. Sử dụng trực tiếp nguyên lý cộng ta có:

- Với i =1 ta cần sử dụng n-1 phép so sánh A[i] với A[j];
- Với i = 2 ta cần sử dụng n-2 phép so sánh A[i] với A[j];
- . . . . .
- Với i = n-1 ta cần sử dụng 1 phép so sánh A[i] với A[j];

Vì vậy tổng số các phép toán cần thực hiện là:

$S = (n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2 \leq n^2 = O(n^2)$.

Ghi chú. Độ phức tạp thuật toán cũng là số lần thực hiện phép toán tích cực. Phép toán tích cực là phép toán thực hiện nhiều nhất đối với thuật toán.

1.4.2. Qui tắc xác định độ phức tạp thuật toán

Qui tắc tổng: Nếu $f_1(x)$ có độ phức tạp là $O(g_1(x))$ và $f_2(x)$ có độ phức tạp là $O(g_2(x))$ thì độ phức tạp của $(f_1(x) + f_2(x)$ là $O(Max(g_1(x), g_2(x))$.

Chứng minh.

- Vì $f_1(x)$ có độ phức tạp là $O(g_1(x)$ nên tồn tại hằng số C_1 và k_1 sao cho $|f_1(x)| \leq |g_1(x)|$ với mọi $x \leq k_1$;
- Vì $f_2(x)$ có độ phức tạp là $O(g_2(x))$ nên tồn tại hằng số C_2 và k_2 sao cho $|f_2(x)| \leq |g_2(x)|$ với mọi $x \leq k_2$;
- Ta lại có :

$$|f_1(x)+ f_2(x)| \leq |f_1(x)| + |f_2(x)|$$
$$\leq C_1|g_1(x)| + C_2|g_2(x)|$$
$$\leq C|g(x)| \text{ với mọi } x > k;$$

Trong đó, $C = C_1 + C_2$; $g(x) = max(g_1(x), g_2(x))$; $k = max (k_1, k_2)$.

Tổng quát. Nếu độ phức tạp của $f_1(x)$, $f_2(x)$,.., $f_m(x)$ lần lượt là $O(g_1(x))$, $O(g_2(x))$,.., $O(g_n(x))$ thì độ phức tạp của $f_1(x) + f_2(x) + ..+f_m(x)$ là $O(max(g_1(x), g_2(x),..,g_m(x))$.

Qui tắc nhân: Nếu f(x) có độ phức tạp là O(g(x) thì độ phức tạp của $f^n(x)$ là $O(g^n(x)$. Trong đó:

$f^n(x)$ = f(x).f(x)….f(x). //n lần f(x).

$g^n(x)$ = g(x).g(x)…g(x).//n lần g(x)

Nói cách khác, đoạn chương trình P có thời gian thực hiện T(n)= O(f(n)). Khi đó, nếu thực hiện k(n) lần đoạn chương trình P với k(n) là O(g(n)) thì độ phức tạp tính toán là O(f(n). g(n)).

Chứng minh. Thật vậy theo giả thiết f(x) là O(g(x)) nên tồn tại hằng số C và k sao cho với mọi x>k thì $|f(x)| \leq C.|g(x)|$. Ta có:

$$\left|f^n(x)\right| = \left|f^1(x).f^2(x)...f^n(x)\right|$$

$$\leq \left|C.g^1(x).C.g^2(x)...C.g^n(x)\right|$$

$$\leq C^n\left|g^n(x)\right| = O\left(g^n(x)\right)$$

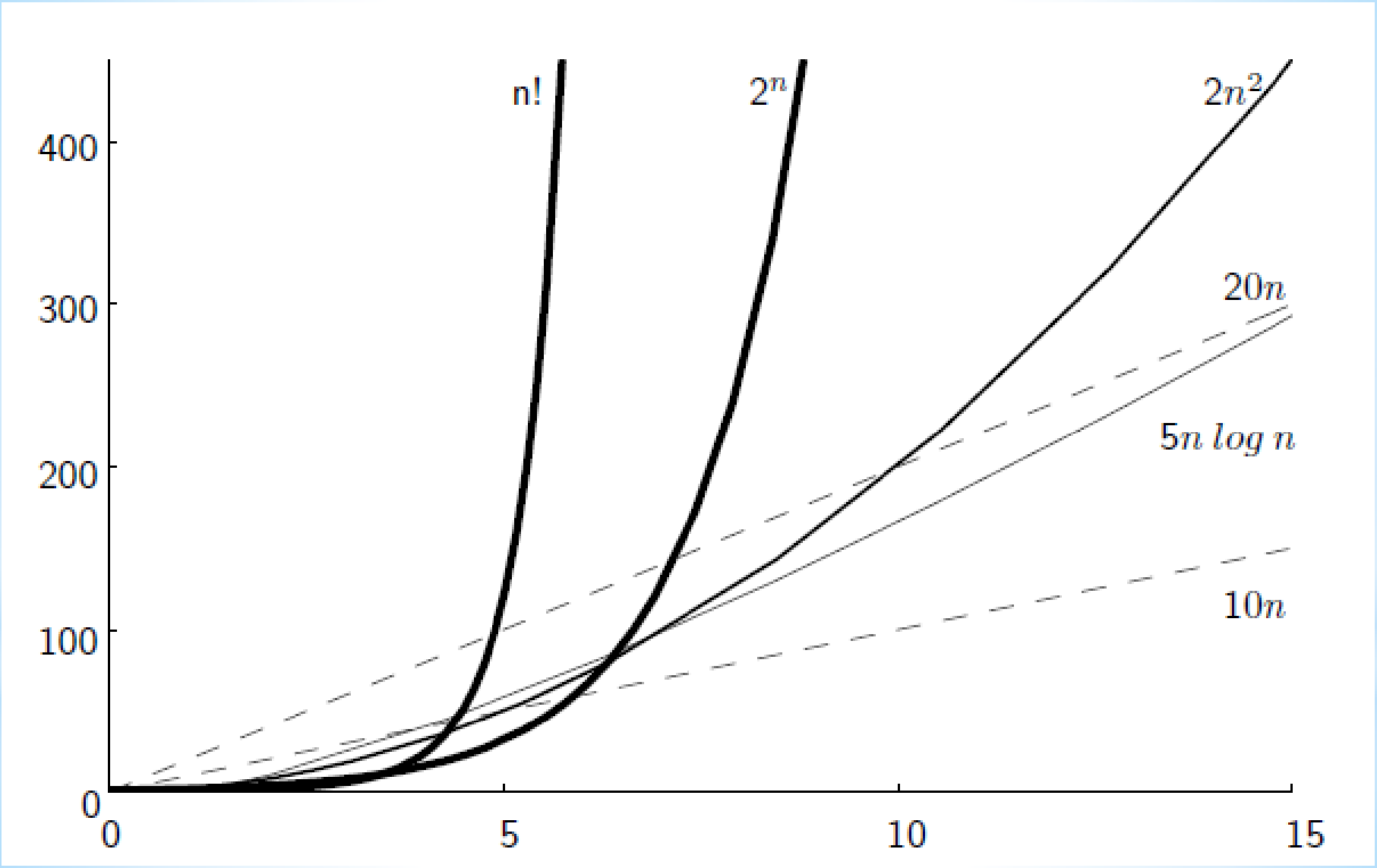
1.4.3. Một số tính chất của độ phức tạp thuật toán:

- Với P(n) là một đa thức bậc k thì $O(P(n)) = O(n^k)$. Vì thế ta nói, một thuật toán có độ phức tạp cấp đa thức là $O(n^k)$.
- Với a, b là hai cơ số tùy ý và f(n) là một hàm xác định dương thì $\log_a f(n) = \log_a b . \log_b (f(n))$. Vì vậy độ phức tạp thuật toán cấp logarit được ký hiệu là O(log(f(n)) mà không cần quan tâm đến cơ số.
- Nếu độ phức tạp thuật toán là hằng số, nghĩa là thời gian tính toán không phụ thuộc vào độ dài dữ liệu được ký hiệu là O(1).
- Một giải thuật có cấp 2^n, n!, n^n được gọi là giải thuật hàm mũ. Những giải thuật này thường có tốc đọ rất chậm.
- Độ phức tạp tính toán của một đoạn chương trình P chính bằng số lần thực hiện một phép toán tích cực. Trong đó, phép toán tích cực trong một đoạn chương trình là phép toán mà số lần thực hiện nó không ít hơn các phép toán khác.

CÁC DẠNG HÀM ĐÁNH GIÁ ĐỘ PHỨC TẠP THUẬT TOÁN

Dạng đánh giá	Tên gọi
O(1)	Hằng số
O(lg lg n)	Log log
O(lg n)	Logarithm
O(n)	Tuyến tính
$O(n^2)$	Bậc hai
$O(n^3)$	Bậc 3
$O(n^m)$	Đa thức
$O(m^n)$	Hàm mũ
O(n!)	Giai thừa

CÁC DẠNG HÀM ĐÁNH GIÁ ĐỘ PHỨC TẠP THUẬT TOÁN



1.4.4. Độ phức tạp của các cấu trúc lệnh

Để đánh giá độ phức tạp của một thuật toán đã được mã hóa thành chương trình máy tính ta thực hiện theo qui tắc sau.

Độ phức tạp hằng số O(1): đoạn chương trình không chứa vòng lặp hoặc lời gọi đệ qui có tham biến là một hằng số.

Ví dụ. Đoạn code dưới đây có độ phức tạp hằng số.

```
for (i=1; i<=c; i++) {
        <Tập các chỉ thị có độ phức tạp O(1)>;
}
```

Độ phức tạp O(n): Độ phức tạp của hàm hoặc đoạn code là O(n) nếu biến trong vòng lặp tăng hoặc giảm bởi mộ hằng số c.

Ví dụ. Đoạn code dưới đây có độ phức tạp hằng số.

```
for (i=1; i<=n; i = i + c ) {
        <Tập các chỉ thị có độ phức tạp O(1)>;
}
for (i=n; i>0; i = i - c ){
        <Tập các chỉ thị có độ phức tạp O(1)>;
}
```

Độ phức tạp hằng số $O(n^c)$: Độ phức tạp của các vòng lặp lồng nhau bằng lũy thừa của số lần lồng nhau.

Ví dụ. Đoạn code dưới đây có độ phức tạp $O(n^2)$.

```
for (i=1; i<=n; i = i + c ) {
        for (j=1; j<=n; j = j + c ){
                <Tập các chỉ thị có độ phức tạp O(1)>;
        }
}
for (i = n; i >0 ; i = i - c ) {
        for (j = i- 1; j>1; j = j -c ){
                <Tập các chỉ thị có độ phức tạp O(1)>;
        }
}
```

Độ phức tạp logarit Log(n): Độ phức tạp của vòng lặp là log(n) nếu biến lặp được chia hoặc nhân với một hằng số c.

Ví dụ. Đoạn code dưới đây có độ phức tạp Log(n).

```
for (i=1; i <=n; i = i *c ){
                <Tập các chỉ thị có độ phức tạp O(1)>;
}
for (j=n; j >0 ; j = j / c ){
                <Tập các chỉ thị có độ phức tạp O(1)>;
}
```

Độ phức tạp hằng số Log Log(n): nếu biến lặp được nhân hoặc chia cho một hàm mũ.

Ví dụ. Đoạn code dưới đây có độ phức tạp Log Log(n).

```
for (i=1; j<=n; j*= Pow(i, c) ){
        <Tập các chỉ thị có độ phức tạp O(1)>;
}
for (j=n; j>=0; j = j- Function(j)  ){
        //Function(j) =sqrt(j) hoặc lớn hơn 2.
        <Tập các chỉ thị có độ phức tạp O(1)>;
}
```

1.5. Cấu trúc dữ liệu (Data Structure)

Cấu trúc dữ liệu được hiểu là phương pháp biểu diễn đối tượng và hành vi của đối tượng ở thế giới thực trong hệ thống máy tính. Dưới đây là một số khái niệm cơ bản về cấu trúc dữ liệu.

1.5.1. Một số thuật ngữ và khái niệm cơ bản

Kiểu (A type): là một tên dùng để chỉ tập các giá trị. Ví dụ Boolean là một tên chỉ tập hai giá trị TRUE, FALSE. Int là một tên dùng để chỉ tập các số nguyên biểu diễn được bằng 2 byte.

Kiểu dữ liệu (A Data Type): là một tên thuộc kiểu **(A Type)** cùng với các thao tác trên kiểu dữ liệu đó. Đối với ngôn ngữ lập trình ta có thể hiểu kiểu dữ liệu như là một biến thuộc kiểu cùng với các phép toán trên nó.

Ví dụ với khai báo:

```
int  a, b;
```

Ta hiểu a và b có dữ liệu kiểu số nguyên (int) là một kiểu dữ liệu. Khi đó, a và b có thể thực hiện được các phép toán số học { +, -, *, /, %, >>, <<, &, |, ^} và các phép toán logic { &&, ||, ! }.

Kiểu trừu tượng(A Abtract Data Type): mô tả thực tế các đối tượng thông qua các kiểu. Như vậy, kiểu dữ liệu trừu tượng có thể hiểu là sự tổ hợp của các kiểu dữ liệu cơ bản. Kiểu dữ liệu trừu tượng được viết tắt là **ADT**.

Cấu trúc dữ liệu (A Data Stucture): là một cài đặt cụ thể của các kiểu dữ liệu trừu tượng. Trong đó, cấu trúc dữ liệu phản ánh không gian nhớ lưu trữ dữ liệu thuộc kiểu.

1.5.2. Phân loại các cấu trúc dữ liệu

Cấu trúc dữ liệu (data structure)	Cấu trúc dữ liệu cơ bản (base data structure)	Số nguyên (integer): short int, long int
		Số thực (real): float, double.
		Ký tự (character): char , wchar_t:
	Cấu trúc dữ liệu ADT (abtract data structure)	Ngăn xếp (stack)
		Hàng đợi (queue): queue, priority queue, circular queue, dqueue.
		Danh sách liên kết đơn: single linked list), circular single linked list.
		Danh sách liên kết kép: double linked list), circular double linked list.
		Cây (tree): Binary Tree, Binary Search Tree, Complete Binary Tree, AVL tree, B-Tree, Red-Black-tree….
		Đồ thị (Graph): Directed Graph, Undirected Graph, Euler Graph, Hamilton Graph, Bipartile Graph, Complete Graph…

1.5. Tính đúng đắn của thuật toán dựa vào dữ liệu

Khi hoàn thành việc xây dựng một thuật toán thì thuật toán đó cần phải được kiểm nghiệm. Phương pháp kiểm nghiệm thuật toán phổ biến hiện nay thường được sử dụng trong tin học là kiểm nghiệm dựa vào dữ liệu. Để có thể kiểm nghiệm thuật toán dựa vào dữ liệu ta cần tiến hành xây dựng các bộ dữ liệu Test. Các bộ dữ liệu Test cần thỏa mãn:

- Kiểm tra được kết quả thực hiện của thuật toán đối với dữ liệu nhỏ.
- Kiểm tra được kết quả thực hiện của thuật toán đối với dữ liệu lớn.
- Kiểm tra được kết quả của thuật toán đối với các trường hợp đặc biệt.
- Kiểm tra được độ phức tạp của thuật toán dựa vào thời gian thực hiện mỗi test.
- Kiểm tra được kỹ thuật cài đặt thuật toán của người lập trình.

Ví dụ 1. Sử dụng cấu trúc dữ liệu thích hợp, hãy viết chương trình chuyển đổi số tự nhiên n thành số ở hệ cơ số b ($2 \le b \le 32$).

Dữ liệu vào (Input) cho bởi file data.in theo khuôn dạng:

- Dòng đầu tiên ghi lại số tự nhiên K là số lượng các test ($k \le 100$).
- K dòng kế tiếp ghi lại mỗi dòng một test. Mỗi test bao gồm một cặp số *n, b*. Hai số được viết cách nhau một vài khoảng trống.

Kết quả ra (Output): ghi lại K dòng trong file ketqua.out, mỗi dòng ghi lại bộ ba số *n, k, x*. Trong đó *x* là số ở hệ cơ số *b* được chuyển đổi từ *n*. Ví dự dưới đây minh họa cho file input và output của bài toán.

Input.in

```
5
8 2
32 16
255 16
100 10
64 32
```

Output.out

```
8 2 1000
32 16 20
255 16 FF
100 10 100
64 32 20
```

Ví dụ 2. Hãy viết chương trình tìm số các số tự nhiên N thỏa mãn đồng thời những điều kiện dưới đây ($N \leq 2^{31}$):

- N là số có K chữ số (K≤15).
- N là số nguyên tố.
- Đảo ngược các chữ số của N cũng là một số nguyên tố.
- Tổng các chữ số của N cũng là một số nguyên tố.
- Mỗi chữ số của N cũng là các số nguyên tố ;
- Thời gian thực hiện chương trình không quá 1sec.

Dữ liệu vào (Input) cho bởi file data.in theo khuôn dạng:

- Dòng đầu tiên ghi lại số tự nhiên M là số lượng các test (M≤100).
- M dòng kế tiếp ghi lại mỗi dòng một test. Mỗi test bao gồm một số *K*. Hai số được viết cách nhau một vài khoảng trống.

Kết quả ra (Output): ghi lại M dòng trong file ketqua.out, mỗi dòng ghi lại bộ hai số số *N, X*. Trong đó *X* là số các số có N chữ số thỏa mãn yêu cầu của bài toán. Ví dự dưới đây minh họa cho file input và output của bài toán.

Input.in	**Output.out**
5	2 0
2	3 8
3	4 15
4	5 46
5	7 359
7	