CHƯƠNG I

NHẬP MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

I. GIẢI THUẬT VÀ CẤU TRÚC DỮ LIỆU

1. Ý nghĩa của cấu trúc dữ liệu

Khi nói tới việc giải quyết bài toán trên máy tính điện tử người ta thường chỉ chú ý tới *giải thuật* (*algorithms*). Đó là dãy các câu lệnh (statements) chặt chẽ và rõ ràng xác địch trình tự các thao tác trên một số đối tượng nào đó sao cho sau một số hữu hạn bước thực hiện ta đạt được kết quả mong muốn.

Nhưng xét cho cùng giải thuật chỉ phản ảnh phép xử lý, còn đối tượng để xử lý trên máy tính chính là *dữ liệu* (*data*). Dữ liệu biểu diễn các thông tin cần thiết của bài toán, các dữ kiện đưa vào các kết quả trung gian ... Không thể nói tới giải thuật mà không nghĩ tới giải thuật đó tác động lên dữ liệu nào. Ngược lại khi nói tới dữ liệu cũng phải hình dung được giải thuật xử lý dữ liệu đó.

Nếu biết tổ chức dữ liệu theo cấu trúc thích hợp thì việc xử lý sẽ thuận lợi và đạt hiệu quả hơn. Với cấu trúc dữ liệu đã chọn ta có giải thuật xử lý tương ứng. Cấu trúc dữ liệu thay đổi, giải thuật cũng thay đổi theo.

◊ *Ví dụ*: Chương trình đặt chỗ máy bay.

Giả sử một hãng hàng không cần chương trình đăng ký chỗ cho một chiếc máy bay 10 chỗ ngồi. Các công việc chính là:

(1) Kiểm tra chỗ trống.

(2) Đăng ký chỗ.

(3) Xoá chỗ đã đăng ký.

Ta xét hai cách tổ chức dữ liệu và giải thuật đếm số chỗ còn trống sau:

a) Thiết kế 1: (vụng)

◊ Trong Pascal:

```
Type SeatStatus = (Occupied,Unoccupied);
Var  Seat1, Seat2, Seat3, Seat4, Seat5, Seat6, Seat7, Seat8, Seat9, Seat10 : SeatStatus;
count: integer;
count:=0; {biến đếm số chỗ trống}
if Seat1=Unoccupied then count:=count+1;
if Seat2=Unoccupied then count:=count+1;
if Seat3=Unoccupied then count:=count+1;
if Seat4=Unoccupied then count:=count+1;
if Seat5=Unoccupied then count:=count+1;
```

```
if Seat6=Unoccupied then count:=count+1;
if Seat7=Unoccupied then count:=count+1;
if Seat8=Unoccupied then count:=count+1;
if Seat9=Unoccupied then count:=count+1;
if Seat10=Unoccupied then count:=count+1;
```

◊ Trong C:

```
#define  MaxSeat  10
enum SeatStatus {Occupied, Unoccupied};
enum SeatStatus   Seat1, Seat2, Seat3, Seat4, Seat5, Seat6, Seat7, Seat8, Seat9,
Seat10;
int count; //biến đếm số chỗ trống
count = 0;
if (Seat1==Unoccupied) count++;
if (Seat2==Unoccupied) count++;
if (Seat3==Unoccupied) count++;
if (Seat4==Unoccupied) count++;
if (Seat5==Unoccupied) count++;
if (Seat6==Unoccupied) count++;
if (Seat7==Unoccupied) count++;
if (Seat8==Unoccupied) count++;
if (Seat9==Unoccupied) count++;
if (Seat10 ==Unoccupied) count++;
```

Ta thấy giải thuật dài dòng và nhàm chán.

b) Thiết kế 2: (tốt hơn)

Trong PASCAL	Trong C
Const MaxSeat = 10; *Type SeatStatus =* *(Occupied,Unoccupied);* *SeatList = array[1..MaxSeat] of SeatStatus;* *Var Seat : SeatList;* *count:=0; {biến đếm số chỗ trống}* *for i:=1 to MaxSeat do* *if Seat[i]=Unoccupied then* *count:=count+1;*	*#define MaxSeat 10* *enum SeatStatus {Occupied, Unoccupied};* *enum SeatStatus Seat[MaxSeat+1];* *count=0; //biến đếm số chỗ trống* *for (i=1; i <=MaxSeat;i++)* *if (Seat[i]==Unoccupied) count++;*

Ta thấy giải thuật ngắn gọn súc tích thể hiện chất trí tuệ cao.

- ***Nguyên lý* (*Niklaus Wirth*)**

Niklous Wirth, cha đẻ của ngôn ngữ lập trình Pascal và kỹ thuật lập trình cấu trúc đã đúc kết ý nghĩa của dữ liệu và mối quan hệ hữu cơ của nó với giải thuật bằng mệnh đề nổi tiếng:

Chương trình = Thuật toán + Cấu trúc dữ liệu

2. Cấu trúc dữ liệu và các vấn đề liên quan

a. Dữ liệu và lưu trữ dữ liệu

- ***Dữ liệu***

Dữ liệu là vật mang thông tin đã được chuẩn hoá.

Chúng ta cần phân biệt dữ liệu với thông tin:

- Dữ liệu tồn tại khách quan.
- Thông tin có ý nghĩa chủ quan.

Trong một bài toán dữ liệu bao gồm một tập các phần tử cơ sở, gọi là *dữ liệu nguyên tử*: nó có thể là một chữ số, một ký tự ..., nhưng cũng có thể là con số, một từ ... Điều đó tuỳ thuộc từng bài toán cụ thể.

Trên cơ sở các dữ liệu nguyên tử, các cách thức liên kết chúng với nhau sẽ dẫn tới các cấu trúc dữ liệu khác nhau.

Trong những năm gần đây lớp các khái niệm về cấu trúc dữ liệu tăng lên đáng kể. Thoạt đầu khi ứng dụng của máy tính mới hạn chế trong phạm vi các bài toán khoa học kỹ thuật thì ta chỉ gặp những cấu trúc dữ liệu đơn giản như số, véc tơ, ma trận, ... Nhưng khi các ứng dụng mở rộng sang các lĩnh vực khác mà ta thường gọi là các *bài toán phi số* với đặc điểm khối lượng dữ liệu lớn, đa dạng, biến động và phép xử lý không chỉ còn là các phép số học, thì những cấu trúc dữ liệu đơn giản trên không còn đủ đặc trưng cho các mối quan hệ mới của dữ liệu nữa.

Đi đôi với các cấu trúc dữ liệu mới cũng xuất hiện các phép toán mới tác động lên các cấu trúc dữ liệu này. Đó là các phép tạo lập hay huỷ bỏ một cấu trúc, phép truy cập vào từng phần tử của cấu trúc, phép bổ sung hay loại bỏ phần tử của cấu trúc.

Các phép toán đó sẽ có tác dụng khác nhau đối với từng cấu trúc. Có phép toán hữu hiệu đối với cấu trúc này nhưng lại tỏ ra không hữu hiệu trên cấu trúc khác.

Vì vậy chọn một cấu trúc dữ liệu phải nghĩ ngay tới các phép toán tác động lên cấu trúc ấy. Và ngược lại, nói tới phép toán lại phải chú ý tới cấu trúc chịu tác động

của phép toán ấy. Cho nên, nói tới cấu trúc dữ liệu là bao hàm luôn cả phép toán tác động lên cấu trúc đó.

Lựa chọn một cấu trúc dữ liệu thích hợp để tổ chức dữ liệu đầu vào và trên cơ sở đó xây dựng giải thuật xử lý hữu hiệu đưa tới kết quả mong muốn cho bài toán là khâu đặc biệt quan trọng.

• ***Cấu trúc lưu trữ***

Cách biểu diễn một cấu trúc dữ liệu trong bộ nhớ được gọi là *cấu trúc lưu trữ* (*storage structure*). Đó chính là cách cài đặt cấu trúc ấy trên máy tính và trên cơ sở các cấu trúc lưu trữ này mà thực hiện các phép xử lý. Sự phân biệt giữa cấu trúc dữ liệu và cấu trúc lưu trữ tương ứng cần được đặt ra. Có thể có nhiều cấu trúc lưu trữ khác nhau cho cùng một cấu trúc dữ liệu, cũng như có thể có những cấu trúc dữ liệu khác nhau mà được cài đặt trong bộ nhớ bởi cùng một kiểu cấu trúc lưu trữ.

Với cấu trúc lưu trữ cũng cần phân biệt *cấu trúc lưu trữ trong* (tương ứng với bộ nhớ trong) với *cấu trúc lưu trữ ngoài* (tương ứng với bộ nhớ ngoài). Chúng có đặc điểm riêng và kéo theo cách xử lý khác nhau.

b. Các kiểu dữ liệu đơn giản

Kiểu dữ liệu là *tập hợp* các dữ liệu cùng tính chất.

i) *Kiểu số nguyên* (*integer*)

♦ *Hệ nhị phân*:

Số nguyên dương trong *hệ nhị phân* biểu diễn như sau:

$$(p_k\, p_{k-1} \ldots p_1\, p_0)_2 \;=\; p_k * 2^k + \ldots + p_1 * 2^1 + p_0 * 2^0$$

trong đó

p_i là *bit* bằng 0 hoặc 1, i=0,…,k.

◊ *Ví dụ*

$$(0110)_2 = 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$$
$$= 0 * 8 + 1 * 4 + 1 * 2 + 0 * 1 = 6$$

Việc đổi số thập phân n sang số nhị phân có thể thực hiện bằng cách chia liên tiếp n cho 2, cho tới khi kết quả là 0. Sau đó ghép các số dư từ dưới lên ta được số nhị phân.

◊ *Ví dụ*: Tìm số nhị phân biểu diễn 14.

14	|	2					
0		7	|	2			
		1		3	|	2	
				1		1	| 2
						1	0 (kết thúc)

Ta ghép các số dư từ dưới lên và nhận được số nhị phân

$$(1110)_2 = 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 14$$

♦ *Cấu trúc lưu trữ*

(1) *Kiểu bù hai* (*two's complement*):

Số nguyên dương lưu trữ như là một dãy các bit biểu diễn nhị phân của số đó.

◊ *Ví dụ:* Giả sử ta dùng 1 byte (8 bit) biểu diễn số 6. Ta đã biết biểu diễn nhị phân của 6 là $(110)_2$. Khi đó số 6 sẽ được lưu trữ trong 1 byte như sau

0	0	0	0	0	1	1	0

bit dấu

↳ Chú ý rằng bit đầu tiên bên trái là bit dấu: số 0 chỉ số không âm.

Số nguyên âm: Biểu diễn của số nguyên âm $-p$, $p > 0$, nhận được bằng cách trước hết tìm dạng nhị phân của p, lấy *phần bù hai* của nó (nghĩa là thay 0 bằng 1 và thay 1 bằng 0), sau đó cộng kết quả với 1.

◊ *Ví dụ:* Biểu diễn bù hai của –6 bằng xâu 8 bit được tính như sau.

Biểu diễn dạng nhị phân của số 6 là

$$(0000\ 0110)_2$$

Lấy phần bù hai của dạng nhị phân trên ta nhận được số

$$(1111\ 1001)_2$$

Cộng số trên với 1 trong hệ nhị phân ta nhận được số

$$(1111\ 1010)_2$$

Kết quả số –6 được lưu trữ trong bằng xâu 8 bit như sau

1	1	1	1	1	0	1	0

bit dấu

↳ Chú ý rằng bit đầu tiên bên trái là bit dấu: số 1 chỉ số âm.

Biểu diễn kiểu bù hai thuận lợi cho việc tính toán, nhưng không thuận lợi cho việc so sánh.

(2) *Kiểu ký pháp lệch* (*biased notation*):

Biểu diễn của số nguyên (dương hoặc âm) p bằng dãy n bit nhận được bằng cách thêm phần lệch 2^{n-1} vào p, và biểu diễn kết quả dạng nhị phân.

◊ *Ví dụ:* Biểu diễn ký pháp lệch của 6 bằng xâu 8 bit được tính như sau.

Thêm phần lệch $2^7 = 128$ vào 6 ta được 134.

Biểu diễn kết quả dạng nhị phân là: $(1000\ 0110)_2$

1	0	0	0	0	1	1	0

bit dấu

↳ Chú ý rằng bit đầu tiên bên trái là bit dấu: số **1** chỉ số không âm.

◊ *Ví dụ*: Biểu diễn ký pháp lệch của -6 bằng xâu 8 bit được tính như sau.

Thêm phần lệch $2^7 = 128$ vào -6 ta được 122.

Biểu diễn kết quả dạng nhị phân: $(0111\ 1010)_2$

Số -6 được lưu trữ trong bằng xâu 8 bit như sau

0	1	1	1	1	0	1	0

bit dấu

↳ Chú ý rằng bit đầu tiên bên trái là bit dấu: số **0** chỉ số âm.

Biểu diễn kiểu ký pháp lệch không thuận lợi cho việc tính toán, nhưng thuận lợi cho việc so sánh.

♦ *Phạm vi* số nguyên p lưu trữ trong xâu n bit:

$$-2^{n-1} \le p \le 2^{n-1} - 1$$

Như vậy

Số nguyên lớn nhất lưu trữ trong 1 byte là $2^7 - 1 = 127$

Số nguyên lớn nhất lưu trữ trong 2 byte là $2^{15} - 1 = 32767$

Số nguyên lớn nhất lưu trữ trong 4 byte là $2^{31} - 1 = 2147483647$

Nếu số lưu trữ nằm ngoài phạm vi lưu trữ thì xảy ra *tràn số* (*overflow*). Vì vậy khi sử dụng kiểu số nguyên cần xác định phạm vi dữ liệu để tránh tràn số.

◊ *Khai báo*

Trong PASCAL	Trong C
var i, j : integer;	*int i, j*;

ii) *Kiểu số thực* (*real*)

- Biểu diễn *nhị phân* số thực có dạng sau

$(p_k\, p_{k-1} ... p_1\, p_0 \bullet p_{-1} p_{-2} ... p_{-l})_2 = p_k * 2^k + ... + p_1 * 2^1 + p_0 * 2^0 + p_{-1} * 2^{-1} + ... + p_{-l} * 2^{-l}$

Dấu **.** ở đây gọi là *dấu chấm nhị phân*, và các vị trí bên phải dấu chấm biểu diễn các số mũ âm của cơ số 2.

◊ *Ví dụ*

Số nhị phân $(110.101)_2$ có giá trị thập phân là

$1*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 4 + 2 + 0 + 1/2 + 0 + 1/8 = 6.625$

• Biểu diễn *dạng khoa học* (scientific) hay dạng *dấu chấm động* (floating point):

Một số nhị phân có thể biểu diễn dưới dạng khoa học sau

$$M * 2^E$$

trong đó số thực nhị phân M là *phần định trị* và số nguyên E là *số mũ*.

◊ *Ví dụ*. Số nhị phân $(110.101)_2$ có thể biểu diễn dưới dạng khoa học như sau

$$0.110101 * 2^3 \quad \text{hoặc} \quad 1.10101 * 2^2$$

♦ *Cấu trúc lưu trữ:* Chuẩn IEEE754/85 dạng 32 bit.

Số thực dạng khoa học hay dấu chấm động được lưu trữ dưới dạng dãy 32 bit:

1 bit	**8 bit**	**23 bit**
s	*e*	*m*

trong đó

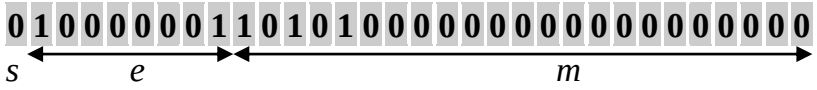
s là bit dấu (số dương *s* = 0, số âm *s* = 1),

e là mã lệch của phần mũ E: *e* = E+127 hay E = *e*–127, số 127 ở đây là độ lệch (bias) hay độ phân cực,

m là phần lẻ của phần định trị M (M = 1.*m*).

Khi đó giá trị của số là: $(-1)^s * (1.m)_2 * 2^{e-127}$.

◊ *Ví dụ*. Lưu trữ số nhị phân $(110.101)_2 = 1.10101 * 2^2$ (= 6.625 dạng thập phân) trong dãy 32 bit từ trái sang phải: bit thứ nhất *s*=0 chỉ số dương, 8 bit tiếp theo là mã lệch *e*=129, kéo theo số mũ E=129-127=2, 23 bit cuối là phần lẻ của phần định trị.



Phạm vi số mũ *E* lưu trữ trong xâu *n* bit là

$$-2^{n-1} \le E \le 2^{n-1} - 1$$

Như vậy, nếu $n = 8$, thì phạm vi số mũ là

từ $-2^7 = -128$ đến $2^7 - 1 = 127$

Tràn số trên (overflow) : Nếu số mũ > số mũ lớn nhất.

Tràn số dưới (underflow) : Nếu số mũ < số mũ bé nhất.

Sai số làm tròn: Nếu phần định trị đòi hỏi nhiều bit hơn số bit được phân phối thì xảy ra sai số.

Ví dụ như biểu diễn cơ số 2 của số 0.7 là

$$(0.1011001100110...)_2$$

trong đó khối 0110 được lặp lại vô hạn lần.

Như vậy với 11 bit, số 0.7 được biểu diễn hoặc

$(0.1011001100)_2 = 0.69921875$ hoặc $(0.1011001101)_2 = 0.700195312$

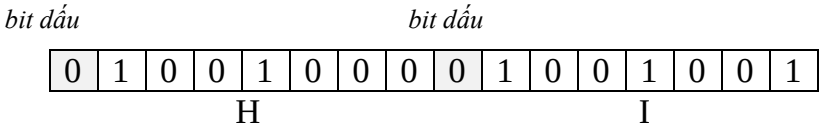
◊ *Khai báo*

Trong PASCAL	Trong C
var r, q : real;	*float r, q;*

iii) *Kiểu ký tự* (*character*)

Máy tính lưu trữ ký tự trên cơ sở gán một mã số cho mỗi ký tự. Sơ đồ chuẩn phổ biến là ASCII (*American Standard Code for Information Interchange*). Mỗi byte biểu diễn 1 ký tự dưới dạng nhị phân.

Ví dụ ký tự H có mã là 72, ký tự I có mã là 73 và xâu ký tự HI được lưu trữ trong từ 2 byte như sau



Phép toán phổ biến nhất đối với dữ liệu ký tự là so sánh hai ký tự hay hai xâu ký tự. Việc so sánh ký tự thực hiện bằng cách so sánh mã của chúng với nhau.

Ví dụ ta có H < I vì mã của H là 72 nhỏ hơn mã của I là 73.

◊ *Khai báo*

Trong PASCAL	Trong C
var c, d : char;	*char c, d;*

iv) *Kiểu Logic* (*boolean*)

Dữ liệu kiệu lôgic chỉ có hai giá trị *đúng* (*true*) và *sai* (*false*). Những giá trị này có thể biểu diễn bằng 1 bit: 0 thể hiện giá trị sai, 1 thể hiện giá trị đúng.

Ngoài ra một số hệ thống khác biểu diễn dữ liệu lôgic bằng 1 byte. Byte với tất cả các bit 0 (0000 0000) là *sai* (*false*), ngược lại thể hiện giá trị *đúng* (*true*).

◊ *Khai báo*

Trong PASCAL	Trong C
var ok : boolean;	*char ok;*

c. Các kiểu dữ liệu cấu trúc

i) *Kiểu mảng*

Mảng là dãy hữu hạn các phần tử cùng kiểu dữ liệu lưu trữ nối tiếp nhau.

Phép toán cơ bản là truy cập trực tiếp đến từng phần tử thông qua địa chỉ.

Trong đa số các ngôn ngữ bậc cao mảng được ký hiệu bởi biến mà giá trị của nó là tập hợp các phần tử tạo nên mảng.

Các phần tử của mảng có thể truy cập thông qua một hay nhiều chỉ số.

Nếu chỉ có một chỉ số thì gọi là mảng một chiều, nếu có nhiều hơn một chỉ số thì gọi là mảng nhiều chiều.

◊ *Khai báo*

Trong PASCAL

```
type  <mảng> = array[<kiểu chỉ số>] of
<kiểu phần tử>;
◊ Ví dụ
const   max  = 20;
        max1 = 3;
        max2 = 4;
type    vecto = array[1..max] of integer;
        matran = array[1..max1,1..max2] of
        real;
var     v,u : vecto;
        A,B : matran;
```

Trong C

```
typedef  <kiểu phần tử>
<mảng>[<max>][<max1>]...;
◊ Ví dụ
#define  max  20
#define  max1 3
#define max2 4
typedef      int  vector[max];
typedef      int  matran[max1][max2];
vector       v,u;
matran       A,B;
```

Địa chỉ từ (word) đầu tiên chứa mảng gọi là *địa chỉ cơ sở*, ký hiệu là *base*(<*mảng*>).

Nếu độ dài của phần tử là *w* từ, thì địa chỉ phần tử thứ *k* của <*mảng*> là

$$base(<mảng>) + (k-1) * w$$

Các mảng 2 chiều có thể coi như mảng 1 chiều của mảng 1 chiều, ví dụ

A,B : array[1..*max*1] *of array*[1..*max*2] *of integer;*

Như vậy địa chỉ của hàng i là:

$$base(<mảng>) + (i-1) * max2 * w$$

và địa chỉ của phần tử chỉ số i,j là

$$base(<mảng>) + (i-1) * max2 * w + (j-1) * w$$

◊ *Mảng nén*: Trường hợp phần tử có độ dài *w* khác bội của từ (word), và được lưu trữ liên tục không để trống byte. Công thức xác định địa chỉ của các phần tử phức tạp hơn.

ii) *Kiểu chuỗi*

Chuỗi là dãy ký tự lấy từ tập hợp các ký tự cho trước, được chứa trong một vùng nhớ liên tục. Có thể coi chuỗi là mảng các ký tự và truy cập đến các ký tự trong chuỗi như các phần tử trong mảng.

◊ *Khai báo*

Trong PASCAL	Trong C
type <kiểu chuỗi> = string[*<max>*]*;* ◊ *Ví dụ* *const max* = 20; *type str = string*[*max*]; *var s,t : str;*	*typedef char <chuỗi>*[*<max>*]*;* ◊ *Ví dụ* *#define max* 20 *typedef char string*[*max*]*;* *string s, t;*

Truy cập ký tự thứ *i* của *s*, *j* của *t* theo chỉ số: *s*[*i*], *t*[*j*] ...

iii) *Kiểu bản ghi* (*record*)

Cấu trúc bản ghi là tập hợp hữu hạn các phần tử, gọi là *trường* (*field*). Khác với mảng các phần tử của bản ghi có thể có kiểu khác nhau.

◊ *Khai báo*

Trong PASCAL	Trong C
type <kiểu bản ghi> = record *<định danh 1>: <kiểu 1>;* *<định danh 2>: <kiểu 2>;* . . . . . . *<định danh n>: <kiểu n>;* *end;*	*struct <cấu trúc> // kiểu dữ liệu* { *<kiểu 1> <định danh 1>;* *<kiểu 2> <định danh 2>;* *...* *<kiểu n> <định danh n>;* }; *struct <cấu trúc> <biến>;*
◊ *Ví dụ* *type Date = record* *Day : 1..31;* *Month : 1..12;* *Year : integer;* *end;* *MarStatus = (Wedding, Divorced, Single);* *NameString = string[20];* *PersonInfo = record* *Name : NameString;* *Birth: Date;* *Status: MarStatus;* *end;* *var NgaySinh : Date;* *Person : PersonInfo;*	◊ *Ví dụ* *struct Date* { *unsigned char Day;* *unsigned char Month;* *unsigned int Year;* }; *typedef char NameString[20];* *enum MarStatus {Wedding, Divorced, Single};* *struct PersonInfo* { *NameString Name;* *struct Date Birth;* *enum MarStatus Status;* }; *struct Date NgaySinh;* *struct PersonInfo Person;*

Cấu trúc lưu trữ bản ghi cũng tương tự, nhưng phức tạp hơn, dữ liệu kiểu mảng.

iv) *Kiểu tập hợp* (*set*)

Tập hợp là nhóm các phần tử khác nhau cùng kiểu không thứ tự.

◊ Khai báo trong PASCAL:

type <kiểu tập hợp> = set of <kiểu phần tử>;

trong đó <kiểu phần tử> có số phần tử hữu hạn (kiểu đoạn con, hay kiểu liệt kê).

◊ *Ví dụ*

```
const max = 200;
type SetOfInt = set of 1..max;
var s1,s2 : SetOfInt;
```

◊ Khai báo trong C:

Giả sử có n phần tử khác nhau a_1, …, a_n có <*kiểu phần tử*>. Có thể biểu diễn tập con của n phần tử a_1, …, a_n bằng mảng như sau:

<kiểu phần tử> element[*n*+1];
typedef char SetOfElement[*n*+1];
SetOfElement s;

trong đó mảng *element*[*n*+1] chứa n phần tử a_1, …, a_n, *elememt*[*i*]=a_i, i=1, …, n. Các biến s biểu diễn tập con của các phần tử a_1, …, a_n, $s[i]$=1 nghĩa là phần tử a_i có trong tập hợp, $s[i]$=0 nghĩa là phần tử a_i không có trong tập hợp.

◊ *Cấu trúc lưu trữ*: Có thể biểu diễn tập hợp trong máy tính bằng dãy các bit, trong đó số bit bằng số phần tử của kiểu phần tử và bit 1 nghĩa là phần tử tương ứng có trong tập hợp, bit 0 nghĩa là phần tử tương ứng không có trong tập hợp.

v) *Kiểu tập tin*

Tập tin là dữ liệu với cấu trúc là dãy (không cố định) các bản ghi lưu trữ ở bộ nhớ ngoài.

Trong PASCAL	Trong C
type <kiểu tập tin>=file of <Kiểu Phần tử>; ◊ *Ví dụ* *type Date = record* *Day :* 1..31; *Month :* 1..12; *Year : integer;* *end;* *MarStatus = (Wedding, Divorced, Single);* *NameString = string*[20]; *PersonInfo = record* *Name : NameString;* *Birth: Date;* *Status: MarStatus;* *end;* *PersonFile = File of PersonInfo;* *var Person : PersonInfo;* *f1,f2 : PersonFile;* *Assign*(*f1, <tên file>*); *Rewrite*(*f1*);{*mở để ghi*} *Write*(*f1, Person*); *Assign*(*f2, <tên file>*);	*FILE *<file>;* ◊ *Ví dụ* *struct Date* { *unsigned char Day;* *unsigned char Month;* *unsigned int Year;* }; *typedef char NameString*[20]; *enum MarStatus* {*Wedding, Divorced, Single*}; *struct PersonInfo* { *NameString Name;* *struct Date Birth;* *enum MarStatus Status;* }; *struct PersonInfo Person;* *FILE *f1, *f2;* *f1 = fopen*(*"PERSON.DAT", "wb"*) *// mở để ghi* *fwrite*(*&Person, sizeof*(*struct PersonInfo*), *1, f1*); *f2 = fopen*(*"PERSON.DAT", "rb"*)

Reset(f2);{mở để đọc} *Read(f2, Person);*	*// mở để đọc* *fread(&Person, sizeof(struct PersonInfo), 1, f2);*

vi) *Kiểu con trỏ* (*pointer*)

Dữ liệu kiểu con trỏ của kiểu phần tử nào đó là địa chỉ của kiểu phần tử đó.

Trong PASCAL	Trong C
type <kiểu con trỏ> = ^ <Kiểu phần tử> ; *◊ Ví dụ* *Type* *InfoType = real;* *pointer = ^element;* *element = record* *Info : InfoType;* *Link : pointer;* *end;* *Var p, First : pointer;* Để truy cập đến các thành phần của biến con trỏ ta sử dụng kí hiệu ^.: *p^.Info, p^.Link*	*typedef <Kiểu phần tử> *<kiểu con trỏ>;* *◊ Ví dụ* *typedef int infotype;* *typedef struct node *nodep;* *struct node {* *infotype data;* *nodep link;* *};* *nodep p, first ;* Để truy cập đến các thành phần của biến con trỏ ta sử dụng kí hiệu ->: *p->data, p->link*

3. Thuật toán

a. Định nghĩa. Thuật toán là tập hợp hữu hạn các thao tác dẫn đến lời giải cho một vấn đề hay bài toán nào đó trong thời gian hữu hạn.

• Các tính chất cơ bản của thuật toán:

- *Tính đúng đắn* : Giải quyết đúng vấn đề.

- *Tính hữu hạn* : Thuật toán phải kết thúc sau một số bước hữu hạn và sau thời gian hữu hạn.

- *Tính phổ quát* : Giải quyết lớp các bài toán (không phải bài toán cụ thể đơn lẻ).

- *Tính tất định* : Tại mỗi bước, với cùng một đầu vào thuật toán phải cho cùng một đầu ra.

- *Tính hiệu quả* : Về không gian (tiết kiệm bộ nhớ), về thời gian (tính nhanh).

b. Cấu trúc điều khiển thuật toán

i) *Cấu trúc tuần tự:*

Trong PASCAL	Trong C
Lệnh gán: <biến> := <biểu thức>;	*Lệnh gán*: <biến> = <biểu thức>;

Ví dụ: *delta* := *b***b* – 4**a***c*; *Thủ tục*: *Ví dụ*: read, write, ... *Lệnh hợp thành* : begin … end;	*Ví dụ*: delta = *b***b* – 4**a***c*; *Thủ tục*: *Ví dụ*: printf, scanf, ... *Lệnh hợp thành* : { … }

ii) *Cấu trúc lặp:*

Trong PASCAL	Trong C
Cấu trúc : *FOR <biến> : = <cận* 1*> TO [DOWNTO] <cận* 2*> DO <công việc>;* Cấu trúc : *WHILE <điều kiện lặp> DO <công việc>;* Cấu trúc : *REPEAT <công việc> UNTIL <điều kiện dừng>;*	Cấu trúc : *for* (*<khởi đầu>; <điều kiện lặp>; <bước nhảy>*) *<công việc>* Cấu trúc : *while* (*<điều kiện lặp>*) *<công việc>* Cấu trúc : *do* *<công việc>* *while* (*<điều kiện lặp>*);

iii) *Cấu trúc chọn:*

Trong PASCAL	Trong C
Cấu trúc : *IF <điều kiện> THEN <công việc* 1*>* [*ELSE <công việc* 2*>*]; Cấu trúc : *CASE <biến> OF* *<trị* 1*> : <công việc* 1*>* *<trị* 2*> : <công việc* 2*>* . . . *<trị n> : <công việc n>* [*ELSE <công việc khác>*] *END;*	Cấu trúc : *if <điều kiện>* *<công việc* 1*>;* [*else <công việc* 2*>;*] Cấu trúc : *switch <biểu thức> // <biểu thức> có giá trị nguyên* { *case <trị* 1*> : <công việc* 1*>; break;* *case <trị* 2*> : <công việc* 2*>; break;* . . . *case <trị n> : <công việc n>; break;* [*default <công việc khác>;*] }

c. Chương trình con

i) *Hàm:* Là chương trình con, xử lý, tính toán với mục đích trả về giá trị của đối tượng nào đó. Cấu trúc hàm như sau.

Trong PASCAL	Trong C
function <tên hàm> (*danh sách tham số*)*: <kiểu dữ liệu hàm>;*	*<kiểu dữ liệu hàm> <tên hàm>* (*danh sách tham số*)

[*khai báo cục bộ*]; *begin* *<các lệnh>;* *<tên hàm>:= <biểu thức>;{lệnh cuối}* *end;*	{ [*khai báo cục bộ*]; *<các lệnh>;* *return <biểu thức>;//lệnh cuối* }

◊ *Chú ý: <kiểu dữ liệu hàm>* được quy định tùy theo ngôn ngữ.

ii) *Thủ tục:* Là chương trình con, xử lý, tính toán một công việc độc lập nào đó. Cấu trúc thủ tục như sau.

Trong PASCAL	Trong C
procedure <tên thủ tục> (*danh sách tham số*); [*khai báo cục bộ*]; *begin* *<các lệnh>;* *end;*	*void <tên thủ tục>* (*danh sách tham số*) { [*khai báo cục bộ*]; *<các lệnh>;* }

II. PHÂN TÍCH THIẾT KẾ CHƯƠNG TRÌNH

1. Vòng đời phát triển phần mềm

Việc giải quyết một vấn đề bằng máy tính đòi hỏi cả phần cứng và phần mềm. *Phần cứng* của một hệ máy tính bao gồm các bộ phận vật lý như *Bộ xử lý trung tâm, bộ nhớ, các thiết bị ngoại vi ...*

Phần mềm liên quan tới các chương trình dùng để điều khiển các phép toán thực hiện bởi phần cứng nhằm mục đích giải quyết vấn đề. Việc xây dựng phần mềm là một quá trình phức tạp. Có thể nói phần mềm vừa là khoa học vừa là nghệ thuật. Là khoa học vì nó sử dụng các kỹ thuật, các phương pháp tiêu chuẩn. Là nghệ thuật vì nó đòi hỏi trí tưởng tượng phong phú, óc sáng tạo thẩm mỹ và sự khéo léo.

Thuật ngữ *Công nghệ phần mềm* (*Software Engineering*) dùng để chỉ việc nghiên cứu và sử dụng các kỹ thuật, phương pháp xây dựng phần mềm.

Vòng đời phát triển phần mềm (*software life cycle*) gồm các bước sau:

(1) Phân tích đặc tả vấn đề.

(2) Chọn lựa cấu trúc dữ liệu và phát triển thuật toán.

(3) Mã hoá chương trình.

(4) Thực hiện và thử chương trình.

(5) Bảo trì chương trình.

a. Phân tích đặc tả vấn đề

Là giai đoạn tìm hiểu mô tả, phân tích vấn đề. Công việc này càng chi tiết chính xác bao nhiêu thì phần mềm càng hiệu quả bấy nhiêu.

Trước hết cần xác định *đầu ra* (*output*), nghĩa là những thông tin nào cần tạo ra để giải quyết vấn đề.

Sau khi đặc tả đầu ra của vấn đề cần phải phân tích tiếp để xác định *đầu vào* (*input*). Đó là những thông tin dùng để giải bài toán. Thông thường những phát biểu ban đầu về bài toán chứa những thông tin không phù hợp, ta cần xác định những thông tin nào có ích cho việc giải bài toán.

Trên cơ sở đầu vào, đầu ra và quy mô bài toán phải đi đến quyết định về phần cứng, phần mềm sử dụng, về trình độ người dùng và các yêu cầu khác.

b. Chọn lựa cấu trúc dữ liệu và phát triển thuật toán

Sau khi đặc tả vấn đề hoàn tất, tiếp đến là giai đoạn thiết kế cấu trúc dữ liệu và giải thuật xử lý dữ liệu để giải quyết bài toán. Đây là giai đoạn khó khăn nhất, đòi hỏi óc sáng tạo, sự khéo léo và kinh nghiệm.

Một trong các phương pháp thiết kế hữu hiệu là *phương pháp thiết kế cấu trúc*.

c. Mã hoá chương trình

Quyết định đầu tiên cần làm khi chuyển thuật toán thành chương trình là sử dụng ngôn ngữ lập trình nào. Có những đặc trương của vấn đề làm việc sử dụng ngôn ngữ này tiện lợi hơn ngôn ngữ khác. Chẳng hạn đối với bài toán khoa học kỹ thuật thì nên sử dụng ngôn ngữ Pascal hay C, đối với bài toán quản lý thì nên dùng một hệ quản trị cơ sở dữ liệu, còn đối với vấn đề liên quan tới hệ chuyên gia hay trí tuệ nhân tạo thì nên dùng Prolog.

Không phụ thuộc vào ngôn ngữ được dùng, các chương trình phải đúng, dễ đọc và dễ hiểu.

Một trong các phương pháp lập trình hữu hiệu là *phương pháp lập trình cấu trúc*.

d. Thực hiện và chạy thử chương trình

Rất hiếm khi một chương trình viết mà không có lỗi. Các lỗi cú pháp (syntax error), chẳng hạn do chấm câu không đúng, các từ khoá bị thiếu ..., và các lỗi khi chạy (run-time error), chẳng hạn như chia cho 0, thường dễ tìm và khắc phục. Tuy nhiên các lỗi lôgic khó tìm hơn nhiều bởi vì chương trình chạy được nhưng không cho kết quả mong muốn. Những lỗi này do mã hoá không chính xác thuật toán, hay do chính bản thân thuật toán không chính xác.

e. Bảo trì chương trình

Sau khi chạy thử và đưa vào sử dụng chương trình sẽ phát huy hiệu quả và được sử dụng trong nhiều năm. Tuy nhiên trong quá trình sử dụng sẽ xuất hiện những vấn đề cần bổ sung, hiệu chỉnh, thậm chí có cả những yêu cầu phát triển chương trình. Đấy là công việc của giai đoạn bảo trì.

Nếu như việc thiết lế và lập trình khoa học, có cấu trúc rõ ràng, và được mô-đun hoá, thì việc bảo trì sẽ thuận lợi hơn nhiều.

Theo các đánh giá thì chi phí cho bảo trì chiếm khoảng 50% tổng chi phí.

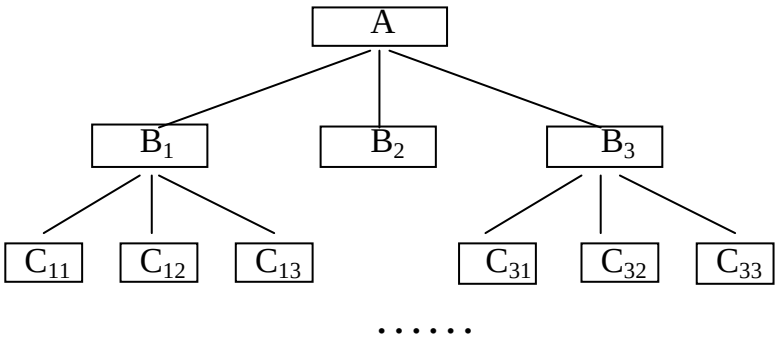
2. Phương pháp thiết kế, lập trình cấu trúc

Phương pháp thiết kế và lập trình cấu trúc gồm hai nội dung chính sau:

a. Thiết kế từ trên xuống và mô-đun hoá

Ý tưởng của phương pháp này xuất phát từ *Chiến thuật chia để trị* để giải quyết một vấn đề. Đó là cách phân tích tổng quát toàn bộ vấn đề. Chia bài toán lớn thành những bài toán nhỏ. Những bài toán nhỏ lại được chia thành những bài toán nhỏ hơn ... Cuối cùng mới đi vào giải quyết chi tiết từng bài toán cụ thể.

Như vậy lời giải của bài toán sẽ được tổ chức theo cấu trúc phân cấp sau: Bài toán lớn A được phân thành các bài toán nhỏ B_1, B_2, B_3, … . Các bài toán B_i lại được phân thành các bài toán nhỏ hơn C_{i1}, C_{i2}, C_{i3}, ... như hình sau



Cách phân tích tổ chức như vậy giúp ta có cái nhìn tổng quát toàn bộ vấn đề, đồng thời có thể phân chia công việc cụ thể một cách chính xác đầy đủ. Mỗi bài toán có thể coi là một mô-đun công việc độc lập.

b. Tinh chỉnh từng bước

Tinh chỉnh từng bước là kỹ thuật thiết kế giải thuật gắn liến với lập trình. Nó phản ánh tinh thần mô-đun hoá bài toán và thiết kế kiểu trên xuống.

Thoạt đầu chương trình thể hiện giải thuật được trình bày bằng ngôn ngữ tự nhiên phản ánh ý chính của công việc. Từ các bước sau những lời những ý đó sẽ được chi tiết hoá dần dần tương ứng với những công việc nhỏ hơn. Ta gọi đó là các bước tinh chỉnh. Sự tinh chỉnh này sẽ hướng tới ngôn ngữ lập trình. Càng ở các bước sau lời lẽ đặc tả công việc xử lý sẽ được thay dần bởi các câu lệnh của ngôn ngữ lập trình.

3. Ví dụ minh hoạ

Bài toán sắp xếp: Sắp xếp một dãy số nguyên bất kỳ theo thứ tự tăng dần.

- *Đặc tả vấn đề*:

  Đầu vào là dãy số nguyên bất kỳ: Nhập từ bàn phím hoặc từ file hoặc tạo ngẫu nhiên.

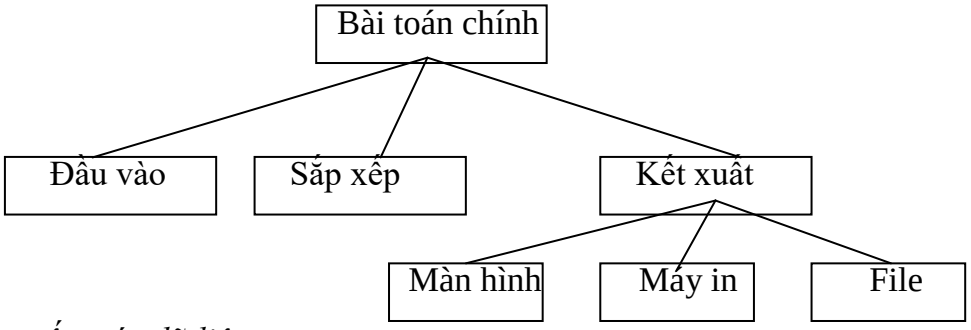
Đầu ra là dãy số nguyên sắp xếp theo thứ tự tăng dần: kết xuất ra màn hình, máy in hoặc file.

Ta sẽ giới hạn trường hợp đầu vào là các số nhập từ bàn phím.

Ý tưởng giải thuật: Chọn số nhỏ nhất đưa vào đầu danh sách đầu ra, chọn số nhỏ thứ nhì đưa vào vị trí thứ hai danh sách đầu ra, chọn số nhỏ thứ ba đưa vào vị trí thứ ba danh sách đầu ra, …

- *Chọn lựa cấu trúc dữ liệu và phát triển thuật toán*:

Ta phân tích bài toán thành các mô-đun như sau:



Chọn lựa cấu trúc dữ liệu:

Nếu số lượng số nguyên đầu vào đã biết trước, ta có thể chọn cấu trúc mảng (danh sách đặc).

Nếu số lượng số nguyên đầu vào bất kỳ, ta có thể chọn cấu trúc danh sách liên kết.

Thiết kế thuật toán:

Thuật toán phụ thuộc vào cấu trúc dữ liệu. Giả sử ta sử dụng cấu trúc mảng gồm *n* phần tử:

- Danh sách chưa sắp xếp : $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$
- Danh sách sắp xếp trung gian : $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$

Khi đó có thể phác thảo thuật toán cho mô-đun *sắp xếp* như sau:

- Đầu vào là dãy ***a***, dãy ***b*** rỗng.
- Từ dãy số chưa được sắp xếp ***a*** chọn ra số nhỏ nhất, đặt nó vào cuối dãy đã sắp xếp ***b***.
- Lặp lại quá trình trên cho đến khi dãy ***a*** rỗng.
- Gán các phần tử của danh sách ***b*** cho các phần tử tương ứng của danh sách ***a***.
- Đầu ra là dãy ***a*** đã dược sắp xếp tăng dần.

- *Tinh chỉnh từng bước*:

Xét thấy tại mỗi bước ta chỉ sử dụng tổng cộng n phần tử của cả mảng ***a*** và ***b***. Vì thế không cần thêm mảng trung gian ***b***, mà sử dụng lần lượt các phần tử của ***a*** để lưu các các số nhỏ nhất được chọn ra.

+ Tinh chỉnh lần 1:

Thủ tục sắp xếp

- Đầu vào : mảng ***a*** có n phần tử (***a*** chưa sắp xếp)
- Đầu ra : mảng ***a*** sắp xếp.

Khai báo biến đếm *i*;

Bắt đầu

Cho i chạy từ 1 đến n-1, với mỗi i thực hiện các công việc:

(i) Tìm số nhỏ nhất a_m trong các số $a_i, a_{i+1}, ..., a_n$.

(ii) Đổi chỗ a_m cho a_i.

Kết thúc

Công việc (i) có thể giải quyết như sau:

Gán i cho m.

Cho j chạy từ i+1 đến n, với mỗi j thực hiện công việc:

Nếu $a_j < a_m$, thì gán j cho m.

Công việc (ii) có thể giải quyết như sau:

Gán a_i cho biến tạm *temp*.

Gán a_m cho a_i.

Gán *temp* cho a_m.

+ Tinh chỉnh lần 2: Đến đây ta có chương trình dưới dạng thủ tục sau

Trong PASCAL

```
procedure SapXep;
var i, j, m, temp:integer;
begin
    for i := 1 to n-1 do
        begin
            m := i;
            for j := i+1 to n do
                if a[j] < a[m] then m := j;
            temp := a[i];
            a[i] := a[m];
            a[m] := temp;
        end;
end;
```

Trong C

```
void SapXep()
{
    int i, j, m, temp;
    for (i=1; i < n; i++)
        {
            m = i;
            for ( j = i + 1; j <= n; j++)
                if (a[j] < a[m]) m = j;
            temp = a[i];
            a[i] = a[m];
            a[m] = temp;
        }
}
```

III. PHÂN TÍCH ĐÁNH GIÁ GIẢI THUẬT

1. Các tiêu chuẩn đánh giá

a. Tính đúng đắn

Thuật giải phải giải đúng bài toán. Thông thường để kiểm tra tính đúng đắn của thuật giải người ta cài đặt chương trình thể hiện thuật giải và chạy thử nghiệm với dữ liệu mẫu và so sánh với kết quả đã biết.

b. Tính hữu hạn

Thuật toán phải kết thúc sau một số bước hữu hạn và sau thời gian hữu hạn (chấp nhận).

c. Tính tất định

Tại mỗi bước, với cùng một đầu vào thuật toán phải cho cùng một đầu ra.

d. Tính phổ quát

Giải quyết lớp các bài toán (không phải bài toán cụ thể đơn lẻ).

e. Tính hiệu quả

- Thời gian thực hiện nhanh.
- Tài nguyên sử dụng tiết kiệm.

2. Phân tích thời gian thực hiện

a. Độ phức tạp tính toán

Với một bài toán, không phải chỉ có giải thuật. Chọn một giải thuật đưa tới kết quả nhanh là một yêu cầu thực tế. Nhưng căn cứ vào đâu để có thể nói giải thuật này nhanh hơn hay chậm hơn giải thuật kia.

Có thể thấy ngay, thời gian thực hiện một giải thuật (chính xác hơn là chương trình thể hiện giải thuật đó) phụ thuộc vào rất nhiều yếu tố. Một yếu tố cần chú ý trước tiên là kích thước dữ liệu đầu vào. Chẳng hạn sắp xếp một dãy số phải chịu ảnh hưởng của số lượng các số thuộc dãy số đó. Nếu gọi n là số lượng dữ liệu (kích thước) đầu vào, thì thời gian thực hiện T của một giải thuật phải được biểu diễn như một hàm của n : $T(n)$.

Các kiểu lệnh và tốc độ xử lý của máy tính, ngôn ngữ lập trình và chương trình dịch ngôn ngữ đó đều ảnh hưởng tới thời gian thực hiện. Nhưng những yếu tố này

không giống nhau trên các máy khác nhau, vì vậy không thể dựa vào chúng để xác lập $T(n)$. Điều đó cũng có nghĩa là $T(n)$ không biểu diễn được bằng giây, phút, ...

Thời gian $T(n)$ ở đây phải hiểu là *cấp độ lớn* của số lượng phép tính, phụ thuộc vào kích thước đầu vào, và được gọi là *độ phức tạp* của giải thuật.

Tuy nhiên, thời gian $T(n)$ không chỉ phụ thuộc vào kích thước đầu vào mà còn phụ thuộc vào trạng thái dữ liệu đầu vào. Chẳng hạn nếu dãy cần sắp xếp đã được sắp xếp từ trước với mức độ nào đó, thì thời gian sắp xếp sẽ nhanh hơn nhiều so với dãy bất kỳ.

Vì vậy chúng ta cần phân biệt các loại thời gian thực hiện chương trình sau:

- Thời gian trung bình : $T_{tb}(n)$
- Thời gian xấu nhất : $T_{max}(n)$
- Thời gian tốt nhất : $T_{min}(n)$

Về lý thuyết, người ta thường đánh giá độ phức tạp $T(n)$ trong trường hợp xấu nhất.

Độ phức tạp thường được biểu diễn thông qua các hàm đa thức, luỹ thừa, hàm mũ. Để so sánh cấp độ lớn giữa các hàm người ta dùng ký hiệu O.

• *Ký pháp O* (*O lớn*): Cho các hàm nguyên $f(n)$ và $g(n)$ phụ thuộc số nguyên n. Ta nói hàm $f(n)$ có *cấp* $g(n)$ và viết

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

nếu tồn tại các hằng số k, n_0 thoả mãn

$$f(n) \le k.g(n), \quad \forall n \ge n_0$$

◊ *Ghi chú*. Nếu tồn tại $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ hữu hạn, thì $f(n) = O(g(n))$.

◊ *Ví dụ*

$f(n) = 2.n^2 - 3.n + 100 \quad \Rightarrow \quad f(n) = O(n^2)$

$f(n) = K$ (hằng số) $\quad \Rightarrow \quad f(n) = O(1)$

$f(n) = 3.n \quad \Rightarrow \quad f(n) = O(2.n) = O(n)$

$f(n) = 2.n + 5.\ln(n) \quad \Rightarrow \quad f(n) = O(n)$

$f(n)$ là đa thức bậc $m \quad \Rightarrow \quad f(n) = O(n^m)$

• *Định nghĩa:* Nếu độ phức tạp T(n) của một thuật toán thoả mãn T(n) = O(g(n)), thì ta nói thuật toán có độ phức tạp g(n).

Thông thường các hàm thể hiện độ phức tạp tính toán của thuật toán có dạng $log_2(log_2 n)$, $log_2(n)$, n, $n.log_2(n)$, n^2, n^3, ..., n^m, 2^n, $n!$, n^n, ...

Các hàm như 2^n, $n!$, n^n gọi là hàm loại *mũ*. Một thuật toán có độ phức tạp hàm mũ thì tốc độ rất chậm. Các hàm $log_2(log_2 n)$, $log_2(n)$, n, $n.log_2(n)$, n^2, n^3, ..., n^m được gọi là *hàm đa thức*. Một thuật toán có độ phức tạp hàm đa thức thì tốc độ chấp nhận được.

Bảng dưới hiển thị giá trị các hàm trên đối với một số giá trị n.

Hàm / ***n***	$log_2(log_2 n)$	$log_2 n$	n	$n.log_2 n$	n^2	n^3	2^n
2	0	1	2	2	4	8	4
4	1	2	4	8	16	64	16
16	2	4	16	64	256	4096	65536
32	$\log_2 5$	5	32	160	1024	32768	2147483684
1048576	4,32	20	1048576	$2{,}1*10^7$	$1{,}1*10^{12}$	$1{,}15*10^{18}$	$6{,}7*10^{315652}$

Giả sử mỗi lệnh thực hiện trong 1 micro giây, bảng dưới chỉ ra thời gian cần thiết để thực hiện f(n) lệnh đối với các hàm thông dụng và với n = 256.

Hàm	***Thời gian***
$\log_2(\log_2 n)$	3 micro giây
$\log_2 n$	8 micro giây
n	0,25 mili giây
$n.\log_2 n$	2 mili giây
n^2	65 mili giây
n^3	17 giây
2^n	$3{,}7*10^{61}$ thế kỷ

b. Các quy tắc tính toán độ phức tạp

• *Quy tắc tổng*

Giả sử $T_1(n) = O(f(n))$ và $T_2(n) = O(g(n))$ là độ phức tạp của 2 đoạn chương trình P_1 và P_2 kế tiếp nhau. Khi đó, độ phức tạp của P_1 và P_2 là

$$T(n) = T_1(n) + T_2(n) = O(f(n) + g(n)) = O(\max(f(n), g(n)))$$

Tổng quát, độ phức tạp của nhiều đoạn chương trình kế tiếp nhau bằng tổng độ phức tạp của mỗi đoạn chương trình:

$T(n) = T_1(n) + T_2(n) + ...+ T_k(n) = O(f_1(n) + f_2(n) + ... + f_k(n))$

$= O(\max(f_1(n), f_2(n), ..., f_k(n)))$

trong đó $T_i(n) = O(f_i(n))$, $\forall i = 1, ..., k$.

◊ *Ví dụ:* Giả sử một chương trình có 3 bước thực hiện mà độ phức tạp từng bước lần lượt là $O(n^2)$, $O(n^3)$ và $O(n.\log_2 n)$.

Khi đó, độ phức tạp chương trình là $O(\max(n^2, n^3, n.\log_2 n)) = O(n^3)$.

• *Quy tắc nhân*

Giả sử $T_1(n) = O(f(n))$ và $T_2(n) = O(g(n))$ là độ phức tạp của 2 đoạn chương trình P_1 và P_2 lồng nhau. Khi đó độ phức tạp của chương trình thực hiện P_1 và P_2 lồng nhau là

$$T(n) = T_1(n) * T_2(n) = O(f(n)*g(n))$$

◊ *Ví dụ*. Câu lệnh gán

Trong PASCAL	Trong C
x := *x*+1;	*x* = *x*+1;

có độ phức tạp là O(1).

Câu lệnh:

Trong PASCAL	Trong C
for i :=1 *to n do x* :=*x*+1;	*for* (*i* = 1; *i* <= *n; i*++) *x*++*;*

có độ phức tạp là $O(n.1) = O(n)$.

Câu lệnh:

Trong PASCAL	Trong C
for i :=1 *to n do* *for j* :=1 *to n do* *x* :=*x*+1;	*for* (*i*=1; *i*<=*n; i*++) *for* (*j* =1; *j*<=*n; j*++) *x* ++*;*

có độ phức tạp là $O(n*n) = O(n^2)$.

◊ *Ví dụ*. Thuật toán tính trị trung bình *Mean* của *n* số.

Trong PASCAL	Trong C
1. Đọc *n*; 2. *Sum* := 0;	1. Đọc *n*; 2. *Sum* = 0;

3. *i* := 1; 4. while (*i* <= *n*) do begin a. đọc *Number*; b. *Sum* := *Sum* + *Number*; c. *i* := *i*+1; end 5. *Mean* := *Sum* / *n*;	3. *i* = 1; 4. while (*i* <= *n*) { a. đọc *Number*; b. *Sum* = *Sum* + *Number*; c. *i*++; } 5. *Mean* = *Sum* / *n*;

Để tính độ phức tạp ta phân tích như sau:

Lệnh	*Số lần thực hiện*
1	1
2	1
3	1
4 (<=)	*n* + 1
a	*n*
b	*n*
c	*n*
5	1
Tổng cộng:	4.*n* + 5

Như vậy độ phức tạp T(*n*) = 4*n* + 5 = O(*n*).

• *Phép toán tích cực*

Dựa vào các nhận xét đã nêu ở trên về các quy tắc tính toán độ phức tạp, ta chỉ cần chú ý tới các bước tương ứng với một phép toán mà ta gọi là phép toán tích cực. *Phép toán tích cực* trong thuật toán là phép toán mà số lần thực hiện nó không ít hơn số lần thực hiện các phép toán khác. Chú ý rằng phép toán tích cực có thể không duy nhất. Độ phức tạp của thuật toán bằng độ phức tạp của phép toán tích cực biểu diễn theo ký pháp O.

Ví dụ trong chương trình tính trị trung bình trên phép toán so sánh trên dòng 4 là tích cực và nó thực hiện *n*+1 lần. Vậy ta có độ phức tạp thuật toán

T(*n*) = O(*n*).

c. Cách tính độ phức tạp (ĐPT) cho các cấu trúc thuật toán

• Lệnh gán: ĐPT bằng kích thước dữ liệu.

Gán 1 hằng: ĐPT = 1.

Gán 1 mảng *n* phần tử: ĐPT = *n*.

• Cấu trúc: IF <*điều kiện*> THEN <*thủ tục*>;

ĐPT = ĐPT(<*điều kiện*>) + ĐPT(<*thủ tục*>)

- Cấu trúc: IF <*điều kiện*> THEN <*thủ tục* 1> ELSE <*thủ tục* 2>;

ĐPT = ĐPT(<*điều kiện*>) + max(ĐPT(<*thủ tục* 1>), ĐPT(<*thủ tục* 2>))

- Cấu trúc lặp: *Với i* chạy từ 1 đến *n thực hiện < thủ tục>*;

ĐPT = *n**ĐPT(<*thủ tục*>)

- Cấu trúc lặp: *Lặp lại < thủ tục> cho đến khi <điều kiện dừng> đúng*;

ĐPT = *n**(ĐPT(<*thủ tục*>) + ĐPT(<*điều kiện dừng*>))

trong đó *n* là số lần tối đa thực hiện vòng lặp.

- Cấu trúc lặp: *Lặp lại < thủ tục> trong khi <điều kiện tiếp tục> đúng*;

ĐPT = *n**(ĐPT(<*thủ tục*>) + ĐPT(<*điều kiện tiếp tục*>))

trong đó *n* là số lần tối đa thực hiện vòng lặp.

- Cấu trúc lặp: *Trong khi <điều kiện lặp> đúng, thực hiện < thủ tục>*;

ĐPT = *n**ĐPT(<*thủ tục*>) + (*n*+1)*ĐPT(<*điều kiện lặp*>)

trong đó *n* là số lần tối đa thực hiện vòng lặp.

◊ *Ví dụ*. Thuật toán tìm kiếm tuyến tính.

Tìm phần tử (biến) *Item* trong danh sách *a*[1], *a*[2], ..., *a*[*n*]. Biến *Found* sẽ có giá trị *true* và biến *Loc* có giá trị là vị trí của *Item* nếu tìm ra, ngược lại *Found = false* và *Loc* = *n*+1.

Trong PASCAL	Trong C
1. *Found := false;* 2. *Loc* := 1; 3. while (*Loc* <= *n*) and not *Found* do 4. if *Item* = *a*[*Loc*] then 5. *Found* := *true* else 6. *Loc* := *Loc* + 1;	*#define false* 0 *#define true* 1 1. *Found = false;* 2. *Loc* = 1; 3. while (*Loc* <= *n* && !*Found*) 4. if (*Item* = = *a*[*Loc*]) 5. *Found = true;* else 6. *Loc*++;

Phép toán (*Loc* <= *n*) là tích cực, thực hiện *n*+1 lần trong trường hợp xấu nhất. Vậy

$$T(n) = O(n).$$

Độ phức tạp tốt nhất: $T_{min}(n) = O(1)$.

Độ phức tạp trung bình: $T_{tb}(n) = n/2 = O(n)$.

◊ *Ví dụ*. Thuật toán tìm kiếm nhị phân.

Tìm phần tử (biến) *Item* trong danh sách có thứ tự tăng dần *a*[1], *a*[2], ..., *a*[*n*]. Biến *Found* sẽ có giá trị *true* và biến *Mid* có giá trị là vị trí của *Item* nếu tìm ra, ngược lại *Found* = *false*. Biến *First* và *Last* dùng để xác định phạm vi tìm kiếm.

Trong PASCAL

```
1. Found := false;
2. First := 1;
3. Last  := n;
4. while (First <= Last) and not Found do
     begin
5.        Mid := (First + Last) div 2;
6.        if Item < a[Mid] then
7.             Last := Mid – 1
          else {Item >= a[Mid]}
8.             if Item > a[Mid] then
9.                  First := Mid + 1
               else
10.                 Found := true
     end;
```

Trong C

```
#define false 0
#define true 1
1. Found = false;
2. First = 1;
3. Last  = n;
4. while ((First <= Last) && (!Found ))
     {
5.        Mid = (First + Last) / 2;
6.        if (Item < a[Mid]) then
7.             Last = Mid – 1;
          else // Item >= a[Mid]
8.             if (Item > a[Mid])
9.                  First = Mid + 1;
               else
10.                 Found = true;
     }
```

Giả sử số bước lặp là *k*. Phép toán 4 (*First* <= *Last*) là tích cực và thực hiện nhiều nhất là *k* lần. Vậy T(*n*) = O(*k*). Ta phải biểu diễn *k* qua *n*.

Mỗi lần qua vòng lặp độ lớn của danh sách giảm một nửa. Lần cuối cùng, tức lần thứ *k*, danh sách còn độ lớn ít nhất là 1. Vì thế độ lớn của danh sách sau (*k*–1) lần lặp là

$$n / 2^{k-1} \geq 2.$$

Suy ra

$$n \geq 2^k \quad \text{hay} \quad k \leq \log_2 n.$$

Như vậy T(*n*) = O(*k*) = O($\log_2 n$).

IV. ĐỆ QUY VÀ GIẢI THUẬT ĐỆ QUY

1. Khái niệm đệ quy

Đệ quy là công cụ hữu hiệu trong khoa học máy tính và toán học để giải quyết nhiều bài toán thực tế. Các ngôn ngữ bậc cao như Pascal, C, ... đều hỗ trợ cài đặt giải thuật đệ quy.

Một đối tượng như thế nào gọi là đối tượng *đệ quy* ? Xét ví dụ sau

◊ *Ví dụ*. Tính $n!$

Nếu $n=0$, thì $0! = 1$.

Nếu $n > 0$, thì $n! = n*(n-1)!$

Như vậy, đối tượng là *đệ quy* nếu nó được định nghĩa bằng chính nó hoặc được xác định với điều kiện nào đó.

◊ *Ví dụ*. Tính ước số chung lớn nhất của hai số nguyên không âm a, b ($a \le b$).

Kí hiệu UCLN(a,b) là ước số chung lớn nhất của a, b. Cách tính như sau:

Nếu $a = 0$, thì UCLN($0,b$) = b.

Nếu $a > 0$, thì UCLN(a,b) = UCLN(b *mod* a, a).

Ví dụ, áp dụng thuật toán đệ quy tìm UCLN của 50 và 70:

= UCLN(50, 70) = UCLN(70 *mod* 50, 50)
= UCLN(20, 50) = UCLN(50 *mod* 20, 20)
= UCLN(10, 20) = UCLN(20 *mod* 10, 10)
= UCLN(0, 10) = 10

• ***Định nghĩa***. Một thuật toán được gọi là *đệ quy* nếu nó giải bài toán bằng cách rút gọn liên tiếp bài toán ban đầu tới bài toán tương tự nhưng có đầu vào nhỏ hơn.

2. Cài đặt thuật toán đệ quy

Trong các ngôn ngữ bậc cao (Pascal, C, ...) thuật toán đệ quy được cài đặt bằng thủ tục hoặc hàm, mà trong thân chương trình thủ tục và hàm lại được gọi lại với tham số đầu vào nhỏ hơn.

◊ *Ví dụ*. Tính $n!$

Trong PASCAL

```
function GiaiThua(n:integer):integer;
begin
    if n =0 then GiaiThua := 1
    else GiaiThua := n*GiaiThua(n-1);
end;
```

Trong C

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

◊ *Ví dụ*. Tính ước số chung lớn nhất của hai số nguyên không âm *a*, *b* ($a \le b$).

Trong PASCAL

```
function UCLN(a, b:integer):integer;
begin
    if a =0 then UCLN := b
    else UCLN := UCLN(b mod a, a);
end;
```

Trong C

```
int  ucln(int a, int b)
{
      if  (a == 0)  return b;
      else  return  ucln(b % a, a);
}
```

◊ *Ví dụ*. Tính số *Fibonacci*.

Dãy số Fibonacci $f(0), f(1), ..., f(n), ...$ được định nghĩa như sau

Nếu $n = 0$, thì $f(0) = 0$.

Nếu $n = 1$, thì $f(1) = 1$.

Nếu $n > 1$, thì $f(n) = f(n-1) + f(n-2)$.

Trong PASCAL

```
function fib(n:integer):integer;
begin
    if n <=1 then fib := n
    else fib := fib(n-1)+fib(n-2);
end;
```

Trong C

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

◊ *Ví dụ. Bài toán tháp Hà Nội*

Bài toán này do *Edouard Lucas* đưa ra ở cuối thế kỷ 19 (Ông cũng là người đưa ra dãy Fibonacci). Bài toán phát biểu như sau. Có 3 cọc, cọc thứ nhất có *n* đĩa kích thước khác nhau xếp chồng nhau, đĩa nhỏ nằm trên đĩa lớn. Hãy chuyển các đĩa từ cọc thứ nhất sang cọc thứ ba, sử dụng cọc trung gian thứ hai, sao cho luôn đảm bảo đĩa nhỏ nằm trên đĩa lớn và mỗi lần chỉ được chuyển 1 đĩa. Tìm phương án di chuyển đĩa tối ưu.

Giả sử các cọc lần lượt kí hiệu là A, B, C và ta phải chuyển *n* đĩa từ cọc A sang cọc C với cọc trung gian B. Thuật toán di chuyển đệ quy như sau:

Nếu $n = 1$, thì chuyển 1 đĩa từ cọc A sang cọc C.

Nếu $n > 1$, thì

Di chuyển $(n-1)$ đĩa từ cọc A sang cọc B với cọc trung gian C;

Chuyển 1 đĩa từ cọc A sang cọc C;

Di chuyển $(n-1)$ đĩa từ cọc B sang cọc C với cọc trung gian A;

Trong PASCAL	Trong C
procedure ThapHaNoi(n:integer; A, B, C :char); *begin* *if n =1 then writeln("Chuyển đĩa từ ", A, " sang " , C)* *else* *begin* *ThapHaNoi(n-1, A, C, B);* *ThapHaNoi(1, A, B, C);* *ThapHaNoi(n-1, B, A, C);* *end;* *end;*	*void thaphanoi(int n, char A, char B, char C)* { *if (n==1)* { *printf("\n%s%c%s%c", "chuyển đĩa từ cọc ", A, " sang cọc", C);* *return;* } *else* { *thaphanoi(n-1, A, C, B);* *thaphanoi(1, A, B, C);* *thaphanoi(n-1, B, A, C);* } }

◊ *Ghi chú:* Số lần di chuyển đĩa là $2^n - 1$. Khi $n = 64$, ta có số lần di chuyển đĩa là:

18 446 744 073 709 551 615

Nếu mỗi giây di chuyển 10 đĩa, thì cần gần 585 triệu thế kỷ mới chuyển xong 64 đĩa.

Nếu mỗi giây di chuyển 10 triệu đĩa, thì cần gần 585 thế kỷ mới chuyển xong 64 đĩa.

BÀI TẬP

1. Viết chương trình đọc từ bàn phím hai số nguyên, rồi hiển thị tổng, hiệu, tích và thương của chúng.
2. Một ngân hàng cho vay với các điều kiện sau: khách hàng phải có thu nhập hàng năm ít nhất là 18 triệu đồng hoặc có tài sản trị giá ít nhất là 100 triệu đồng. Ngoài ra khách hàng không được có số nợ vượt quá 36 triệu đồng. Viết chương trình đọc thu nhập, tài sản và số nợ (kiểu số nguyên) của một khách hàng, rồi hiển thị một trong hai thông báo sau: "Cho vay" hoặc "Không cho vay".
3. Viết chương trình chuyển đổi số giây sang dạng giờ phút giây (ví dụ 3812 giây đổi thành 1 giờ 3 phút 32 giây).
4. Viết chương trình đọc ba số nguyên (biểu diễn chiều dài ba cạnh tam giác), rồi hiển thị một trong các thông báo sau: "Không phải tam giác", "Tam giác đều", "Tam giác vuông", "Tam giác cân" (không đều, không vuông), "Tam giác thường" (không cân, không vuông).
5. Viết chương trình đọc số nguyên, rồi hiển thị số đảo ngược (ví dụ 547 đảo thành 745).
6. Viết chương trình đọc số nguyên, rồi hiển thị tổng các chữ số của nó.
7. Viết chương trình hiển thị tất cả các số tự nhiên có 3 chữ số sao cho tổng các chữ số bằng tích của chúng (ví dụ 123 có 1+2+3=1*2*3=6).
8. Viết chương trình nhập số tự nhiên, rồi hiển thị một trong các thông báo sau: "Số nguyên tố", "Không phải số nguyên tố".
9. Số hoàn thiện là số tự nhiên có tổng các ước số (kể cả 1 và nhỏ hơn chính nó) bằng chính nó, ví dụ 6 là số hoàn thiện vì 6 = 1 + 2 + 3. Viết chương trình nhập số tự nhiên, rồi hiển thị một trong các thông báo sau: "Số hoàn thiện", "Không phải số hoàn thiện".
10. Một đường thẳng có thể được biểu diễn bởi phương trình :

    $ax + by + c = 0$ (a, b, c là các số thực, a và b không đồng thời bằng 0)

    Viết chương trình đọc hai bộ ba số thực (tức 6 số thực) trên hai hàng tương ứng với hai phương trình của đường thẳng trên mặt phẳng. Chương trình hiển thị thông báo về mối quan hệ giữa hai đường thẳng ấy. Các thông báo có thể là :

"Hai đường thẳng cắt nhau, nhưng không vuông góc",

"Hai đường thẳng vuông góc",

"Hai đường thẳng song song, nhưng không trùng nhau",

"Hai đường thẳng trùng nhau".

Hướng dẫn:

Cho hai đường thẳng d1: a1*x + b1*y + c1 = 0 và d2: a2*x + b2*y + c2 = 0.

```
if  (a1*b2 – a2*b1 != 0)
    if (a1*a2+b1*b2 == 0) "vuông góc";
    else "cắt nhau, nhưng không vuông góc";
else
    if ((a1*c2 – a2*c1 == 0) && (b1*c2 – b2*c1 == 0)) "trùng nhau";
    else "song song, nhưng không trùng nhau";
```

11. Viết chương trình để tính diện tích các hình tròn, hình chữ nhật, và hình tam giác như sau:

a. Hình tròn: Nhập một số dương, chương trình hiển thị diện tích hình tròn có bán kính là số vừa nhập.

b. Hình chữ nhật: Nhập hai số dương trên cùng một hàng, chương trình hiển thị diện tích hình chữ nhật có chiều dài và rộng là hai số vừa nhập.

c. Tam giác: Nhập ba số dương trên cùng một hàng, chương trình hiển thị diện tích hình tam giác có chiều dài ba cạnh là ba số vừa nhập.

Dùng ba hàm để tính diện tích các hình trên.

12. Một chuỗi được gọi là *palindrome* nếu đảo ngược các ký tự của nó ta nhận được chuỗi ban đầu, ví dụ từ *MADAM* là palindrome. Viết chương trình nhập một chuỗi, rồi hiển thị một trong các thông báo sau: "palindrome", "Không phải palindrome".

13. Hai chuỗi là *anagram* của nhau nếu một chuỗi là hoán vị các ký tự của chuỗi kia, ví dụ *dear* và *read* là anagram của nhau. Viết chương trình nhập hai chuỗi, rồi hiển thị một trong các thông báo sau: "anagram", "không phải anagram".

14. Viết chương trình kiểm tra mảng số nguyên có được sắp xếp tăng dần không.

15. Viết chương trình kiểm tra mảng số nguyên có đối xứng không.

16. Viết chương trình tìm phần tử lớn nhất và nhỏ nhất của mảng số nguyên, hiển thị khoảng cách giữa hai phần tử đó.

17. Viết chương trình tạo 2 mảng A, B mỗi mảng chứa 30 số tự nhiên ngẫu nhiên nhỏ hơn 100. Sau đó hiển thị hợp, giao và hiệu của chúng (coi mảng là tập hợp).

18. Có 4 nhân viên bán 5 sản phẩm. Viết chương trình nhập mảng giá sản phẩm, mảng lương cơ bản của nhân viên và ma trận số lượng bán từng mặt hàng của mỗi nhân viên. Sau đó

a) Tính tổng số tiền bán được cho mỗi nhân viên.

b) Tính tổng số tiền hoa hồng bằng 5% doanh số bán hàng cho mỗi nhân viên.

c) Tính tổng lương có bản và hoa hồng cho mỗi nhân viên.

19. Viết chương trình đọc tập tin văn bản rồi hiển thị bảng tần suất các ký tự từ A đến Z (không phân biệt chữ thường và chữ hoa) xuất hiện trong văn bản.

20. Viết chương trình tạo lập danh sách 10 người bao gồm các thông tin *Họ tên, Ngày sinh, Nơi sinh*.

21. Viết chương trình nhập vào 2 số nguyên và dùng thủ tục đệ quy tìm ước số chung lớn nhất của chúng.

22. Viết chương trình nhập số nguyên n và dùng thủ tục đệ quy tính số Fibonacci thứ n.

23. Viết chương trình (đệ quy) giải bài toán tháp Hà Nội.

24. Viết chương trình nhập số tự nhiên $n > 0$ từ bàn phím và dùng thủ tục đệ quy tính $1 + 1/2 + 1/3 + ... + 1/n$.

25. Viết chương trình nhập số tự nhiên $n > 0$ từ bàn phím và dùng thủ tục đệ quy tính $1 + 1/2! + 1/3! + ... + 1/n!$.

26. Viết chương trình nhập số tự nhiên $n > 0$ từ bàn phím và dùng thủ tục đệ quy tính $n!!$ ($n!! = 1.3.5. \dots .(n-2).n$ nếu n lẻ, $n!! = 2.4.6. \dots . (n-2).n$ nếu n chẵn).

27. Viết chương trình nhập số tự nhiên $n > 0$ từ bàn phím và dùng thủ tục đệ quy liệt kê tất cả các chuỗi nhị phân độ dài n.

28. Viết chương trình nhập số tự nhiên $n > 0$ từ bàn phím và dùng thủ tục đệ quy liệt kê tất cả các hoán vị của n số $1, 2, \dots, n$.

29. Viết chương trình nhập số tự nhiên $n > 0$ từ bàn phím và dùng thủ tục đệ quy chuyển sang dạng nhị phân.

30. Viết chương trình nhập số tự nhiên $n > 0$ từ bàn phím và dùng thủ tục đệ quy hiển thị các thừa số nguyên tố của n theo thứ tự giảm dần.

31. Viết chương trình nhập số tự nhiên $n > 0$ từ bàn phím và dùng thủ tục đệ quy để hiển thị n dòng đầu tiên của tam giác Pascal. Tam giác Pascal $a_{i,j}$, $0 \le j \le i$, có dạng như sau

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
….

Trong đó: $a_{i,0} = a_{i,i} = 1 \; \forall i$, $a_{i,j} = a_{i-1,j-1} + a_{i-1,j} \; \forall 1 < j < i$.

32. Viết chương trình xếp 8 quân hậu trên bàn cờ vua sao cho các quân hậu không ăn nhau.

33. *Bài toán mã đi tuần*: Viết chương trình tìm đường đi qua tất cả các ô trên bàn cờ vua của quân ngựa sao cho không có ô nào đi qua 2 lần.