



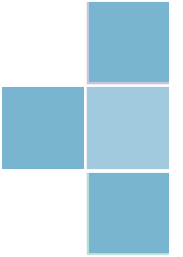
KHOA CÔNG NGHỆ THÔNG TIN – ĐẠI HỌC ĐÀ LẠT



Chương 1
**Một số vấn đề cơ bản của cấu trúc dữ liệu
và thuật giải**



Nội dung



1. Từ bài giải đến chương trình
2. Kiểu dữ liệu trừu tượng
3. Phân tích thuật giải
4. Một số lược đồ thuật giải



1. Từ bài giải đến chương trình



1.1. Mô hình bài toán thực tế

1.2. Thuật giải



1.1. Mô hình bài toán thực tế



- ❖ Để giải một bài giải trong thực tế bằng máy tính ta phải bắt đầu từ việc xác định bài giải.
- ❖ Nhiều thời gian và công sức bỏ ra để xác định bài giải cần giải quyết, tức là phải trả lời rõ ràng câu hỏi "phải làm gì?" sau đó là "làm như thế nào?"
- ❖ Để giảm bớt sự phức tạp của bài giải thực tế, ta phải hình thức hóa nó, nghĩa là phát biểu lại bài giải thực tế thành một bài giải hình thức.



Ví dụ



❖ Tô màu bản đồ thế giới với số màu sử dụng ít nhất:

■ Yêu cầu:

- Mỗi nước đều được tô một màu,
- Hai nước láng giềng (cùng biên giới) thì phải được tô bằng hai màu khác nhau.

■ Mô hình bài toán:

- Mỗi đỉnh xem là 1 nước.
- 2 nước láng giềng được nối bằng 1 cạnh.
- Mỗi đỉnh đều phải được tô màu, hai đỉnh có cạnh nối thì phải tô bằng hai màu khác nhau và ta cần tìm một phương án tô màu sao cho số màu được sử dụng là ít nhất.



1.2. Thuật giải



- ❖ Knuth (1973) định nghĩa thuật giải là một chuỗi hữu hạn các thao tác để giải một bài giải nào đó Các tính chất quan trọng của thuật giải là:
 - Hữu hạn (finiteness): thuật giải phải luôn luôn kết thúc sau một số hữu hạn bước.
 - Xác định (definiteness): mỗi bước của thuật giải phải được xác định rõ ràng và phải được thực hiện chính xác, nhất quán.
 - Hiệu quả (effectiveness): các thao tác trong thuật giải phải được thực hiện trong một lượng thời gian hữu hạn.
- ❖ Ngoài ra thuật toán còn có:
 - Đầu vào và đầu ra.



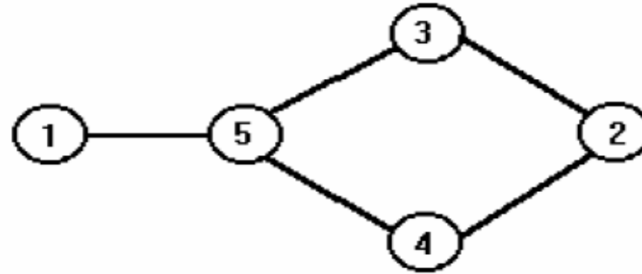
Ví dụ



- ❖ Tô màu bản đồ thế giới với số màu sử dụng ít nhất → bài toán tô màu đồ thị.
 - Sử dụng phương pháp vét cạn (thử tất cả các trường hợp có thể có).
- ❖ Một giải pháp như thế gọi là một HEURISTIC. HEURISTIC cho bài giải tô màu đồ thị, thường gọi là thuật giải "háu ăn" (GREEDY) là:
 - Chọn một đỉnh chưa tô màu và tô nó bằng một màu mới C nào đó.
 - Duyệt danh sách các đỉnh chưa tô màu. Đối với một đỉnh chưa tô màu, xác định xem nó có kề với một đỉnh nào được tô bằng màu C đó không. Nếu không có, tô nó bằng màu C đó.

Ví dụ

❖ Tô đồ thị sau



Tô theo GREEDY
(xét lần lượt theo số thứ tự các đỉnh)

1: đỏ; 2: đỏ

3: xanh; 4: xanh

5: vàng

Tối ưu
(thử tất cả các khả năng)

1,3,4 : đỏ

2,5 : xanh



Các bước tiếp cận với một bài giải



1. Mô hình hoá bài giải bằng một mô hình giải thích hợp.
2. Tìm thuật giải trên mô hình này.
3. Phải hình thức hoá thuật giải bằng cách viết một thủ tục bằng ngôn ngữ giả, rồi chi tiết hoá dần ("mịn hoá").
4. Cài đặt thuật giải trong một ngôn ngữ lập trình cụ thể (Pascal,C,...)



2. Kiểu dữ liệu trừu tượng



❖ *Khái niệm trừu tượng hóa:*

- Trong tin học, trừu tượng hóa nghĩa là đơn giản hóa, làm cho nó sáng sủa hơn và dễ hiểu hơn.

❖ *Trừu tượng hóa chương trình*

- Trừu tượng hóa chương trình là sự định nghĩa các chương trình con để tạo ra các phép giải trừu tượng.
- **Ví dụ:** Nhập, Xu_ly, Xuất là các phép giải trừu tượng

```
void Main()
```

```
{
```

```
    Nhập( Lop);
```

```
    Xu_ly (Lop);
```

```
    Xuất (Lop);
```

```
}
```



2. Kiểu dữ liệu trừu tượng



❖ *Trừu tượng hóa dữ liệu*

- Trừu tượng hóa dữ liệu là định nghĩa các kiểu dữ liệu trừu tượng.
- Một kiểu dữ liệu trừu tượng (Abstract Data Type - ADT) là một mô hình giải học cùng với một tập hợp các phép giải (operator) trừu tượng được định nghĩa trên mô hình đó.
- ADT là sự tổng quát hoá của các kiểu dữ liệu nguyên thuỷ.



2. Kiểu dữ liệu trừu tượng



- Ví dụ: một danh sách (LIST) các số nguyên và các phép giải trên danh sách là:
 - Tạo một danh sách rỗng.
 - Lấy phần tử đầu tiên trong danh sách và trả về giá trị null nếu danh sách rỗng.
 - Lấy phần tử kế tiếp trong danh sách và trả về giá trị null nếu không còn phần tử kế tiếp.
 - Thêm một số nguyên vào danh sách.

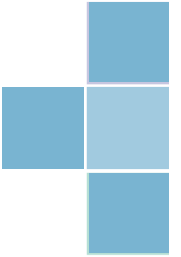
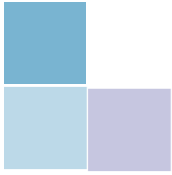


2. Kiểu dữ liệu trừu tượng



❖ *Kiểu dữ liệu, cấu trúc dữ liệu và kiểu dữ liệu trừu tượng*

- Kiểu dữ liệu là một tập hợp các giá trị và một tập hợp các phép giải trên các giá trị đó.
 - Ví dụ kiểu Boolean (TRUE, FALSE) và các phép giải trên nó như OR, AND, NOT,... Kiểu Integer là tập hợp các số nguyên có giá trị từ -32768 đến 32767 cùng các phép giải cộng, trừ, nhân, chia, Div, Mod...
- Kiểu dữ liệu có 2 loại:
 - Kiểu dữ liệu sơ cấp là kiểu dữ liệu mà giá trị dữ liệu của nó là đơn nhất. Ví dụ: kiểu Boolean, Integer....
 - Kiểu dữ liệu có cấu trúc hay còn gọi là cấu trúc dữ liệu là kiểu dữ liệu mà giá trị dữ liệu của nó là sự kết hợp của các giá trị khác. Ví dụ: ARRAY là một cấu trúc dữ liệu.



Các kiểu dữ liệu C/C++

1. Các kiểu dữ liệu cơ sở:
 - Ký tự: char
 - Số nguyên : int, unsigned, long
 - Số thực: float, double
2. Các cấu trúc dữ liệu cơ bản
 - Mảng 1 chiều: ***type ten_mang[MAX];***
 - Mảng 2 chiều: ***type ten_mang[MAX][MAX]***
 - Xâu ký tự (chuỗi): ***char ten_chuoi[MAX];***
 - Kiểu Cấu trúc:

```
struct Ten_Kieu  
{  
  
    //các trường dữ liệu  
  
};
```

...



3. Phân tích thuật giải

- 3.1. Thuật giải và các vấn đề liên quan
- 3.2. Tính hiệu quả của thuật giải
- 3.3. Ký hiệu O và biểu diễn thời gian chạy bởi ký hiệu O
- 3.4. Đánh giá thời gian chạy của thuật giải



3.1. Thuật giải và các vấn đề liên quan



- ❖ Thuật giải được hiểu là sự đặc tả chính xác một dãy các bước thực hiện giải quyết một vấn đề.
- ❖ Mỗi thuật giải có một dữ liệu vào (Input) và một dữ liệu ra (Output);
- ❖ Khi trình bày một thuật giải để cho ngắn gọn nhưng vẫn đảm bảo đủ chính xác, người ta thường biểu diễn thuật giải dưới dạng giả mã (pseudo code).



3.2. Tính hiệu quả của thuật giải



- ❖ Lựa chọn thuật giải để áp dụng dựa vào các tiêu chí sau:
 - Thuật giải đơn giản, dễ hiểu.
 - Thuật giải dễ cài đặt (dễ viết chương trình)
 - Thuật giải cần ít bộ nhớ
 - Thuật giải chạy nhanh

3.3. Ký hiệu O và biểu diễn thời gian chạy bởi ký hiệu O

❖ Định nghĩa ký hiệu O

- Giả sử $f(n)$ và $g(n)$ là các hàm thực không âm của đối số nguyên không âm n . Ta nói “ $f(n)$ là ô lớn của $g(n)$ ” và viết là $f(n) = O(g(n))$ nếu tồn tại các hằng số dương c và n_0 sao cho $f(n) \leq cg(n)$ với mọi $n \geq n_0$.

❖ Ví dụ 1.6: Giả sử $f(n) = 5n^3 + 2n^2 + 13n + 6$,

- ta có: $f(n) = 5n^3 + 2n^2 + 13n + 6 \leq 5n^3 + 2n^3 + 13n^3 + 6n^3 = 26n^3$
- Bất đẳng thức trên đúng với mọi $n \geq 1$, và ta có $n_0 = 1$, $c = 26$.
- Do đó, ta có thể nói $f(n) = O(n^3)$.

❖ Tổng quát nếu $f(n)$ là một đa thức bậc k của n :

- $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ thì $f(n) = O(n^k)$



Biểu diễn thời gian chạy của thuật giải



- ❖ Thời gian chạy của thuật giải là một hàm của cỡ dữ liệu vào: hàm $T(n)$
- ❖ $T(n) = O(f(n))$, biểu diễn này có nghĩa là thời gian chạy $T(n)$ bị chặn trên bởi hàm $f(n)$.
- ❖ **Định nghĩa.** Ta nói $f(n)$ là cận trên chặn của $T(n)$ nếu
 - $T(n) = O(f(n))$, và
 - Nếu $T(n) = O(g(n))$ thì $f(n) = O(g(n))$.
- ❖ Nếu $T(n) = O(1)$ thì điều này có nghĩa là thời gian chạy của thuật giải bị chặn trên bởi một hằng số nào đó.
- ❖ . Nếu $T(n) = O(n)$, thì thời gian chạy của thuật giải bị chặn trên bởi hàm tuyến tính

3.4. Biểu diễn thời gian chạy của thuật giải

Kí hiệu	Tên gọi
$O(1)$	hằng
$O(\log n)$	logarit
$O(n)$	tuyến tính
$O(n \log n)$	$n \log n$
$O(n^2)$	bình phương
$O(n^3)$	lập phương
$O(2^n)$	mũ

Để so sánh thời gian chạy của các thuật giải thời gian đa thức và các thuật giải thời gian mũ, chúng ta hãy xem xét bảng sau:

Thời gian chạy	Cỡ dữ liệu vào					
	10	20	30	40	50	60
N	0,00001 giây	0,00002 giây	0,00003 giây	0,00004 giây	0,00005 giây	0,00006 giây
N²	0,0001 giây	0,0004 giây	0,0009 giây	0,0016 giây	0,0025 giây	0,0036 giây
N³	0,001 giây	0,008 giây	0,027 giây	0,064 giây	0,125 giây	0,216 giây
N⁵	0,1 giây	3,2 giây	24,3 giây	1,7 phút	5,2 phút	13 phút
2ⁿ	0,001 giây	1,0 giây	17,9 phút	12,7 ngày	35,7 năm	366 thế kỷ
3ⁿ	0,059 giây	58 phút	6,5 năm	3855 thế kỷ	2.10⁸ thế kỷ	1,3.10¹³ thế kỷ

Ta giả thiết rằng mỗi phép giải sơ cấp cần 1 micro giây để thực hiện. Thuật giải có thời gian chạy n^2 , với cỡ dữ liệu vào $n = 20$, nó đòi hỏi thời gian chạy là $20^2 \times 10^{-6} = 0,004$ giây

3.4. Đánh giá thời gian chạy của thuật giải

❖ Luật tổng

- Giả sử thuật giải gồm hai phần (hoặc nhiều phần), thời gian chạy của phần đầu là $T_1(n)$, phần sau là $T_2(n)$. Khi đó thời gian chạy của thuật giải là $T_1(n) + T_2(n)$ sẽ được suy ra từ sự đánh giá của $T_1(n)$ và $T_2(n)$ theo luật sau:
 - Giả sử $T_1(n) = O(f(n))$ và $T_2(n) = O(g(n))$. Nếu hàm $f(n)$ tăng nhanh hơn hàm $g(n)$, tức là $g(n) = O(f(n))$, thì $T_1(n) + T_2(n) = O(f(n))$.
- Ví dụ:
 - Giả sử thuật giải gồm ba phần, thời gian chạy của từng phần được đánh giá là $T_1(n) = O(n \log n)$, $T_2(n) = O(n^2)$ và $T_3(n) = O(n)$.
 - Khi đó thời gian chạy của toàn bộ thuật giải là $T(n) = T_1(n) + T_2(n) + T_3(n) = O(n^2)$, vì hàm n^2 tăng nhanh hơn các hàm $n \log n$ và n .



Đánh giá thời gian chạy của thuật giải



❖ Thời gian chạy của các lệnh

- Thời gian thực hiện các phép giải sơ cấp là $O(1)$.
- **Lệnh gán**
- **Lệnh lựa chọn**
- **Các lệnh lặp**
- **Phân tích các hàm đệ quy**



Lệnh gán, lựa chọn



❖ **Lệnh gán:** $X = \langle \text{biểu thức} \rangle$ có thời gian thực hiện $O(1)$.

❖ **Lệnh lựa chọn:** Lệnh lựa chọn **if-else** có dạng

if ($\langle \text{điều kiện} \rangle$)

lệnh 1

else

lệnh 2

- Giả sử thời gian đánh giá $\langle \text{điều kiện} \rangle$ là $T_0(n)$, thời gian thực hiện lệnh 1 là $T_1(n)$, thời gian thực hiện lệnh 2 là $T_2(n)$.
- Thời gian thực hiện lệnh lựa chọn if-else sẽ là thời gian lớn nhất trong các thời gian $T_0(n) + T_1(n)$ và $T_0(n) + T_2(n)$.
- Thời gian chạy của lệnh lựa chọn switch được đánh giá tương tự như lệnh if-else, chỉ cần lưu ý rằng, lệnh if-else có hai khả năng lựa chọn, còn lệnh switch có thể có nhiều hơn hai khả năng lựa chọn.



Các lệnh lặp: for, while, do-while



- Cần đánh giá số tối đa các lần lặp, giả sử đó là $L(n)$.
- Giả sử thời gian thực hiện thân lệnh lặp ở lần thứ i ($i=1,2,\dots, L(n)$) là $T_i(n)$.
- Mỗi lần lặp, chúng ta cần kiểm tra điều kiện lặp, giả sử thời gian kiểm tra là $T_0(n)$. Như vậy thời gian chạy của lệnh lặp là:
$$\sum_{i=1}^{L(n)} (T_0(n) + T_i(n))$$
- Công đoạn khó nhất trong đánh giá thời gian chạy của một lệnh lặp là đánh giá số lần lặp.



Các lệnh lặp: for, while, do-while



❖ Trường hợp hay gặp là:

- Kiểm tra điều kiện lặp (thông thường là đánh giá một biểu thức) chỉ cần thời gian $O(1)$, thời gian thực hiện các lần lặp là như nhau và giả sử ta đánh giá được là $O(f(n))$;
- Khi đó, nếu đánh giá được số lần lặp là $O(g(n))$, thì thời gian chạy của lệnh lặp là $O(g(n)f(n))$.

Các lệnh lặp: for, while, do-while

❖ Ví dụ: Giả sử ta có mảng A các số thực, cỡ n và ta cần tìm xem mảng có chứa số thực x không. Điều đó có thể thực hiện bởi thuật giải tìm kiếm tuần tự như sau:

(1) $i = 0$;

(2) while ($i < n \ \&\& \ x \neq A[i]$)

(3) $i++$;

- Lệnh (1) có thời gian chạy là $O(1)$.
- Lệnh lặp (2)-(3) có số tối đa các lần lặp là n, đó là trường hợp x chỉ xuất hiện ở thành phần cuối cùng của mảng $A[n-1]$ hoặc x không có trong mảng.
- Thân của lệnh lặp là lệnh (3) có thời gian chạy $O(1)$.
- Do đó, lệnh lặp có thời gian chạy là $O(n)$. Thuật giải gồm lệnh gán và lệnh lặp với thời gian là $O(1)$ và $O(n)$, nên thời gian chạy của nó là $O(n)$.

Các lệnh lặp: for, while, do-while

❖ **Ví dụ:** Thuật giải tạo ra ma trận đơn vị A cấp n;

(1) for (i = 0 ; i < n ; i++)

(2) for (j = 0 ; j < n ; j++)

(3) A[i][j] = 0;

(4) for (i = 0 ; i < n ; i++)

(5) A[i][i] = 1;

- Lệnh lặp for đầu tiên (các dòng (1)-(3)) có thân lại là một lệnh lặp for ((2)-(3)).
- Số lần lặp của lệnh for ((2)-(3)) là n, thân của nó là lệnh (3) có thời gian chạy là $O(1)$, do đó thời gian chạy của lệnh lặp for này là $O(n)$.
- Lệnh lặp for ((1)-(3)) cũng có số lần lặp là n, thân của nó có thời gian đã đánh giá là $O(n)$, nên thời gian của lệnh lặp for ((1)-(3)) là $O(n^2)$.
- Tương tự lệnh for ((4)-(5)) có thời gian chạy là $O(n)$.
- Sử dụng luật tổng, ta suy ra thời gian chạy của thuật giải là $O(n^2)$.



Phân tích các hàm đệ quy



- ❖ Các hàm đệ quy là các hàm có chứa lời gọi hàm đến chính nó.
- ❖ Giả sử ta có hàm đệ quy F , thời gian chạy của hàm này là $T(n)$, với n là cỡ dữ liệu vào. Khi đó thời gian chạy của các lời gọi hàm ở trong hàm F sẽ là $T(m)$ với $m < n$. Trước hết ta cần đánh giá thời gian chạy của hàm F trên dữ liệu cỡ nhỏ nhất $n = 1$, giả sử $T(1) = a$ với a là một hằng số nào đó.
- ❖ Sau đó bằng cách đánh giá thời gian chạy của các câu lệnh trong thân của hàm F , chúng ta sẽ tìm ra quan hệ đệ quy biểu diễn thời gian chạy của hàm F thông qua lời gọi hàm, tức là biểu diễn $T(n)$ thông qua các $T(m)$, với $m < n$.

Phân tích các hàm đệ quy

- ❖ Chẳng hạn, giả sử hàm đệ quy F chứa hai lời gọi hàm với thời gian chạy tương ứng là $T(m_1)$ và $T(m_2)$, trong đó $m_1, m_2 < n$, khi đó ta thu được quan hệ đệ quy có dạng như sau:

$$T(1) = 1$$

$$T(n) = f(T(m_1), T(m_2))$$

- ❖ Trong đó, f là một biểu thức nào đó của $T(m_1)$ và $T(m_2)$. Giải quan hệ đệ quy trên, chúng ta sẽ đánh giá được thời gian chạy $T(n)$. Nhưng cần lưu ý rằng, giải các quan hệ đệ quy là rất khó khăn, chúng ta sẽ đưa ra kỹ thuật giải cho một số trường hợp đặc biệt.

Phân tích các hàm đệ quy

❖ Ví dụ: (Hàm tính giai thừa của số nguyên dương n).

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

- Giả sử thời gian chạy của hàm là $T(n)$, với $n = 1$ ta có $T(1) = O(1)$. Với $n > 1$, ta cần kiểm tra điều kiện của lệnh if-else và thực hiện phép nhân n với kết quả của lời gọi hàm, do đó $T(n) = T(n-1) + O(1)$. Như vậy ta có quan hệ đệ quy sau:
 - $T(1) = O(1)$
 - $T(n) = T(n-1) + O(1)$ với $n > 1$
- Thay các ký hiệu $O(1)$ bởi các hằng số dương a và b tương ứng, ta có
 - $T(1) = a$
 - $T(n) = T(n-1) + b$ với $n > 1$
- Sử dụng các phép thế $T(n-1) = T(n-2) + b$, $T(n-2) = T(n-3) + b, \dots$, ta có
 - $T(n) = T(n-1) + b$
 - $\quad = T(n-2) + 2b$
 - $\quad = T(n-3) + 3b$
 - $\quad \dots$
 - $\quad = T(1) + (n-1)b$
 - $\quad = a + (n-1)b$
- Từ đó, ta suy ra $T(n) = O(n)$.



4. Một số lược đồ thuật giải



Nhắc lại thuật giải đệ quy:

Recursion() \equiv

{

 if(điều kiện dừng)

 Xử lý trường hợp đặc biệt;

 else

 if (điều kiện hợp lệ)

 Xử lý đệ qui;

 //Liên hệ đệ qui

}

4.1. Thuật giải vét cạn

4.2. Thuật giải chia để trị (Divide - and - conquer)

4.3. Thuật giải quay lui



4.1. Thuật toán vét cạn



- ❖ Ý tưởng thuật giải vét cạn, còn gọi là thuật giải Brute-force, là xem xét tất cả các ứng viên để phát hiện đối tượng mong muốn. Thuật giải vét cạn sinh ra tất cả các khả năng có thể có và kiểm tra mỗi khả năng có thỏa yêu cầu bài giải không?
- ❖ Thuật giải vét cạn thường dùng để giải các bài giải liệt kê tổ hợp, hoán vị.
- ❖ Thuật giải vét cạn kém hiệu quả vì độ phức tạp tính giải lớn, nhưng cũng có nhiều vấn đề không có các giải quyết nào khác bằng vét cạn.
- ❖ Một số thuật giải quan trọng liên quan đến vét cạn như : thuật giải sinh, quay lui, nhánh cận, . . .

4.2. Thuật giải chia để trị

- ❖ Nếu gọi $D\&C(\mathcal{R})$ - Với \mathcal{R} là miền dữ liệu - là hàm thể hiện cách giải bài giải theo phương pháp chia để trị thì ta có thể viết :

```
void D&C( $\mathcal{R}$ )
{
    If ( $\mathcal{R}$  đủ nhỏ)
        giải bài giải;
    Else
    {
        Chia  $\mathcal{R}$  thành  $\mathcal{R}_1, \dots, \mathcal{R}_m$ ;
        for ( $i = 1; i \leq m; i++$ )
            D&C( $\mathcal{R}_i$ );
        Tổng hợp kết quả;
    }
}
```



4.3. Thuật giải quay lui



- ❖ Lược đồ phương pháp quay lui có thể viết bằng thủ tục sau, với n là số bước cần phải thực hiện, k là số khả năng mà x_i có thể chọn lựa.

Try(i) \equiv

for ($j = 1 \rightarrow k$)

 If (x_i chấp nhận được khả năng j)

 {

 Xác định x_i theo khả năng j ;

 Ghi nhận trạng thái mới;

 if($i < n$)

 Try($i+1$);

 else

 Ghi nhận nghiệm;

 Trả lại trạng thái cũ cho bài giải;

 }



Bài tập



❖ Bài 1. Bài toán tìm min, max

Tìm giá trị min, max trong đoạn $a[1..r]$ của mảng $a[0...n-1]$.

❖ Bài 2. (Bài toán hoán đổi 2 phần trong 1 dãy.)

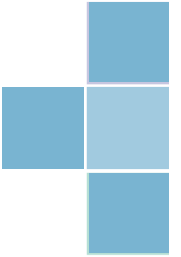
- $a[1..n]$ là một mảng gồm n phần tử. Ta cần chuyển m ($1 \leq m \leq n$) phần tử đầu tiên của mảng với phần còn lại của mảng ($n-m$ phần tử) mà không dùng một mảng phụ .
- Chẳng hạn, với $n = 8$, $a[8] = (1, 2, 3, 4, 5, 6, 7, 8)$
 - Nếu $m = 3$, thì kết quả là : $(4, 5, 6, 7, 8, 1, 2, 3)$
 - Nếu $m = 5$, thì kết quả là : $(6, 7, 8, 1, 2, 3, 4, 5)$
 - Nếu $m = 4$, thì kết quả là : $(5, 6, 7, 8, 1, 2, 3, 4)$



Bài tập



- ❖ Bài 3: Sử dụng thuật toán quay lui, viết chương trình tùy chọn thực hiện các thao tác :
 - Liệt kê các dãy nhị phân độ dài n
 - Liệt kê các tổ hợp chập k trong tập n số nguyên dương đầu tiên.
 - Liệt kê các hoán vị
- ❖ Bài 4. Bài toán Ngựa đi tuần.
 - Cho bàn cờ có $n \times n$ ô. Một con ngựa được phép đi theo luật cờ vua, đầu tiên được đặt ở ô có tọa độ x_0, y_0 .
 - Vấn đề là hãy chỉ ra tất cả hành trình (nếu có) của ngựa – Đó là ngựa đi qua tất cả các ô của bàn cờ, mỗi ô đi qua đúng một lần.



Bài tập về nhà

❖ $n=8$, $a[]$

12 8 15 0 1 6 20 10

Mô tả các bước thực hiện bằng tay cho thuật toán:

MinMax(a , 0, 7, min, max)

Bài toán tìm min, max

```
void MinMax(int a[MAX], int l, int r, int &min, int &max)
{
    int min1, min2, max1, max2;
    if (l == r)
    {
        min = a[l];
        max = a[l];
    }
    else
    {
        MinMax(a, l, (l + r) / 2, min1, max1);
        MinMax(a, (l + r) / 2 + 1, r, min2, max2);
        if (min1 < min2)
            min = min1;
        else
            min = min2;
        if (max1 > max2)
            max = max1;
        else
            max = max2;
    }
}
```




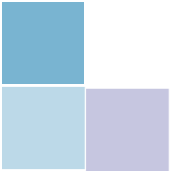
Bài 2. (Bài toán hoán đổi 2 phần trong 1 dãy)



- ❖ Nếu $m = n - m$: Hoán đổi các phần tử của 2 nửa mảng có độ dài bằng nhau :
- ❖ Nếu $m \neq n - m$:
 - Nếu $m < n - m$: hoán đổi m phần tử đầu với m phần tử cuối của phần còn lại. Sau đó trong mảng $a[1..n-m]$ ta chỉ cần hoán đổi m phần tử đầu với phần còn lại.
 - Nếu $m > n - m$: hoán đổi $n-m$ phần tử đầu tiên với $n-m$ phần tử sau. Sau đó trong mảng $a[n-m+1 .. n]$ ta hoán đổi $n-m$ phần tử cuối mảng với các phần tử của phần đầu.

Bài 2. (Bài toán hoán đổi 2 phần trong 1 dãy)

```
void Exchange(int a[MAX], int i, int j, int m)
{
    for (int k = 0; k < m; k++)
    {
        HoanVi(a[i + k], a[j + k]);
    }
}
```



```
void Transpose(int a[MAX], int n, int m)
{
    int i, j;
    i = m;
    j = n-m;
    m = m + 1;
    while (i!=j)
    {
        if (i > j)
        {
            Exchange(a, m - i-1, m-1, j);
            i = i - j;
        }
        else
        {
            j = j - i;
            Exchange(a, m - i-1, m + j-1, i);
        }
    }
    Exchange(a, m - i-1, m-1, i);
}
```

Liệt kê các dãy nhị phân độ dài n

```
int n=3,k;
int a[MAX], //luu tru day nhii phan, hoan vi, to hop
b[MAX]; //danh dau
int dem=0; //dem day nhii phan, hoan vi, to hop
void Xuat_KQ(int a[MAX], int n)
{
    int i;
    cout << "kq" << setw(3) << dem << " : ";
    for (i = 1; i <= n; i++)
        cout << setw(2) << a[i];
    cout << endl;
}
```

```
void LietKe_DayNP(int i)//Try
{
    int j;
    for (j = 0; j <= 1; j++)
    {
        a[i] = j;
        if (i < n)
            LietKe_DayNP(i + 1);
        else
        {
            dem++;
            Xuat_KQ(a, n);
        }
    }
}
```

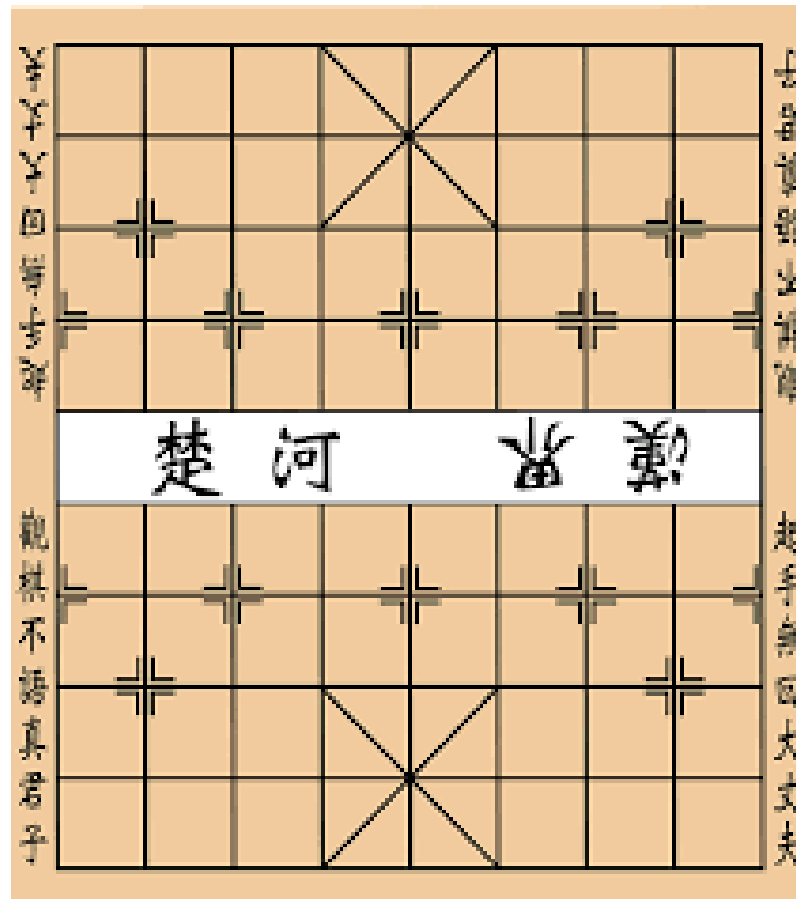
Bài toán Ngựa đi tuần

```
int a[8] = { 2, 1, -1, -2, -2, -1, 1, 2 };  
int b[8] = { 1, 2, 2, 1, -1, -2, -2, -1 };
```

x,y

u=x+a[i]

v=y+b[i]



```
void Try(int i, int x, int y, int h[MAX][MAX], int n, int &slg)
{
    int k, u, v;
    for (k = 0; k <= 7; k++)
    {
        u = x + a[k];
        v = y + b[k];
        if (1 <= u && u <= n && 1 <= v && v <= n && h[u][v] == 0)
        {
            h[u][v] = i;
            if (i < n*n)
                Try(i + 1, u, v, h, n, slg);
            else
            {
                slg++;
                cout << "\nLoi giai thu " << slg << " :";
                Xuat_HanhTrinh(h, n);
            }
            h[u][v] = 0;
        }
    }
}
```

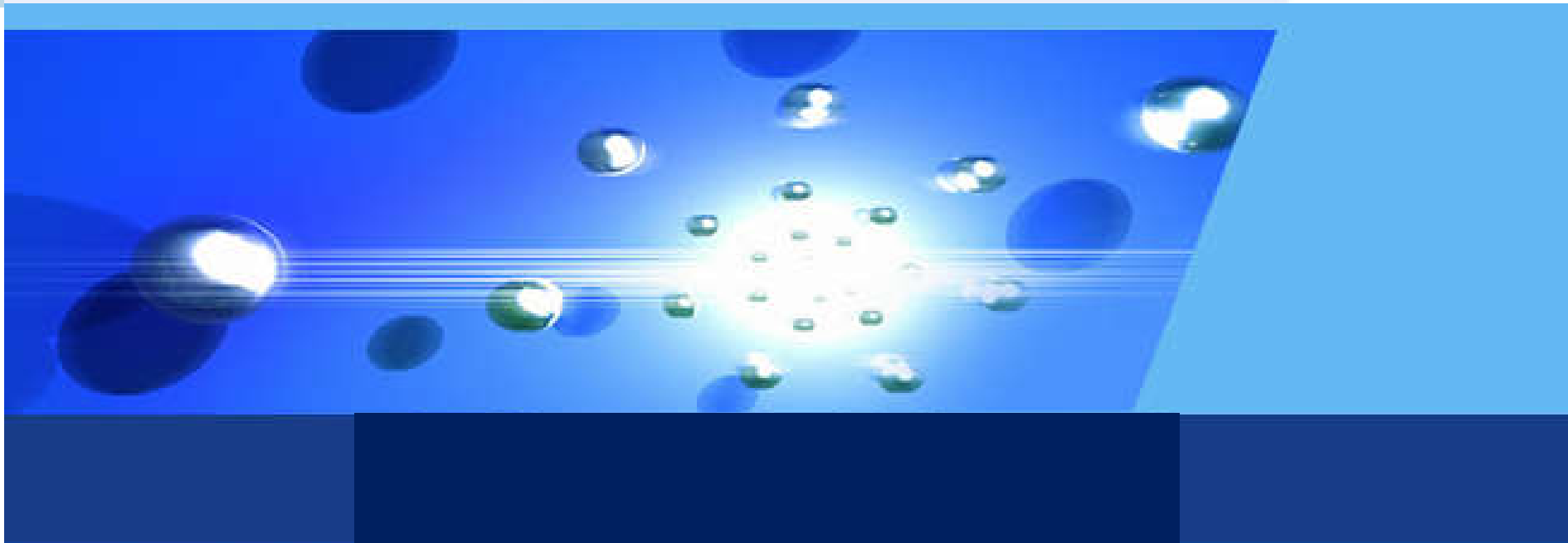
//Xua ma tran h : hanh trinh cua ngua

void Xuat_HanhTrinh(int h[MAX][MAX], int n)

```
{
    int i, j;
    cout << setiosflags(ios::left);
    for (i = 1; i <= n; i++)
    {
        cout << endl << endl;
        for (j = 1; j <= n; j++)
        {
            cout << setw(5)<<h[i][j];
        }
        cout << endl << endl;
    }
}
```

void Init(int h[MAX][MAX], int n)

```
{
    int i, j;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            h[i][j] = 0;
}
```



Hết chương 1