CHƯƠNG 1
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

GV. Ngô Công Thắng

Bộ môn Công nghệ phần mềm Khoa Công nghệ thông tin

Website: dse.vnua.edu.vn/ncthang

1. Mối quan hệ giữa cấu trúc dữ liệu và giải thuật

1.1. Giải thuật (thuật toán, algorithms)

- Khái niệm: Giải thuật là một hệ thống các thao tác, các phép toán được thực hiện theo trình tự nhất định trên một số đối tượng dữ liệu nào đó, sao cho sau một số bước hữu hạn ta có được kết quả mong muốn.
- Giải thuật phản ánh các phép xử lý, còn đối tượng xử lý là dữ liệu.

1.1. Giải thuật (thuật toán, algorithms)

- Giải thuật phải có các tính chất cơ bản sau:
  - Tính thực hiện được:
  - Tính kết thúc:
  - Tính kết quả: Phải cho kết quả mong muốn.
  - Tính hiệu quả:
  - Tính duy nhất:
  - Tính tổng quát: Phải áp dụng cho mọi bài toán cùng loại.

1.2. Cấu trúc dữ liệu

- Khái niệm dữ liệu: Dữ liệu là các phần tử biểu diễn các thông tin cần thiết cho bài toán.
- Một bài toán có thể có các loại dữ liệu: Dữ liệu vào, dữ liệu trung gian, dữ liệu ra.
  - Dữ liệu vào là dữ liệu cần đưa vào để xử lý, đây chính là đầu vào của bài toán.
  - Dữ liệu trung gian là dữ liệu chứa các kết quả trung gian trong quá trình xử lý.
  - Dữ liệu ra là dữ liệu chứa kết quả mong muốn của bài toán.
- Giải thuật thực hiện biến đổi từ các dữ liệu vào thành các dữ liệu ra.

1.2. Cấu trúc dữ liệu (*tiếp*)

- Ví dụ 1: Ta xét bài toán tính học bổng cho sinh viên theo chế độ hiện hành. Các dữ liệu của bài toán bao gồm:
  - Dữ liệu vào: Họ và tên, Điểm các môn, Số trình các môn học.
  - Dữ liệu trung gian: Điểm trung bình
  - Dữ liệu ra: Học bổng

1.2. Cấu trúc dữ liệu (*tiếp*)

- Ví dụ 2: Xét bài toán giải phương trình bậc hai $ax2 + bx + c = 0$. Các dữ liệu của bài toán này như sau:
  - Dữ liệu vào: a, b, c
  - Dữ liệu trung gian: delta
  - Dữ liệu ra: x1, x2

1.2. Cấu trúc dữ liệu (*tiếp*)

- Dữ liệu nguyên tử là phần tử dữ liệu cơ sở không thể tách nhỏ ra được, có thể là một chữ số, một kí tự, một giá trị logic,... Trong một bài toán, dữ liệu bao gồm một tập các dữ liệu nguyên tử.
- Từ các dữ liệu nguyên tử ta có thể tạo thành các cấu trúc dữ liệu bằng các cách thức liên kết khác. Chẳng hạn liên kết các kí tự lại với nhau tạo thành cấu trúc dữ liệu kiểu xâu kí tự, liên kết các số lại với nhau theo kiểu một dãy số ta được cấu trúc dữ liệu kiểu mảng một chiều.

1.2. Cấu trúc dữ liệu (*tiếp*)

- Tóm lại, Cấu trúc dữ liệu là cách tổ chức các phần tử dữ liệu của bài toán.

1.2. Cấu trúc dữ liệu (*tiếp*)

- Khái niệm về Cấu trúc lưu trữ: Cách biểu diễn một cấu trúc dữ liệu trong bộ nhớ được gọi là cấu trúc lưu trữ, đó chính là cách cài đặt cấu trúc dữ liệu trên máy vi tính.
  - Có thể có nhiều cấu trúc lưu trữ khác nhau cho một cấu trúc dữ liệu. Chẳng hạn một cấu trúc dữ liệu kiểu mảng ta có thể lưu trữ bằng các ô nhớ kế tiếp nhau trong bộ nhớ hoặc có thể lưu trữ bằng các ô nhớ không kế tiếp nhau trong bộ nhớ.
  - Có thể có nhiều cấu trúc dữ liệu khác nhau được cài đặt trong bộ nhớ bằng một cấu trúc lưu trữ. Chẳng hạn cấu trúc xâu kí tự, cấu trúc mảng đều có thể cài đặt trong bộ bằng các ô kế tiếp nhau.

1.2. Cấu trúc dữ liệu (*tiếp*)

- Mỗi một ngôn ngữ lập trình đều có các cấu trúc dữ liệu tiền định (định sẵn), bởi vậy khi chọn ngôn ngữ lập trình nào thì ta phải chấp nhận cấu trúc dữ liệu tiền định của nó, phải vận dụng linh hoạt các cấu trúc dữ liệu này vào bài toán cần giải quyết.

1.3. Mối quan hệ giữa cấu trúc dữ liệu và giải thuật

- Xét tới giải thuật thì phải xét giải thuật đó tác động trên cấu trúc dữ liệu nào.
- Xét tới cấu trúc dữ liệu thì phải hiểu cấu trúc dữ liệu đó cần được tác động bằng giải thuật gì để được kết quả mong muốn.
- **Cấu trúc dữ liệu nào thì giải thuật đó. Khi cấu trúc dữ liệu thay đổi giải thuật cũng thay đổi theo.**
- Mối quan hệ giữa cấu trúc dữ liệu và giải thuật được Niklaus Wirth tổng kết như sau:

Cấu trúc dữ liệu + Giải thuật = Chương trình

2. Các cách diễn đạt giải thuật

2.1. Liệt kê các bước bằng lời

- Trong cách diễn đạt này ta phải viết từng bước làm công việc gì: Bước 1, Bước 2….

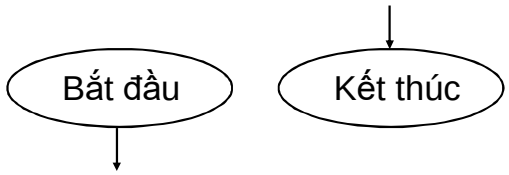
2. Các cách diễn đạt giải thuật

2.2. Lưu đồ giải thuật

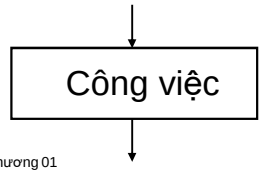
- Lưu đồ giải thuật là một sơ đồ có hướng diễn đạt các bước thực hiện của giải thuật.
- Lưu đồ giải thuật giúp người lập trình xem xét sự làm việc của giải thuật khá chi tiết và cụ thể.
- Lưu đồ giải thuật bao gồm các hình cơ bản nối với nhau bởi các đường có hướng.

2.1. Lưu đồ giải thuật (*tiếp*)

- Các hình cơ bản trong lưu đồ giải thuật gồm có:
  - Hình elíp thể hiện sự bắt đầu và kết thúc của giải thuật.

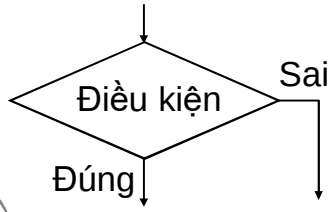


  - Hình chữ nhật chỉ các thao tác, công việc cần thực hiện.

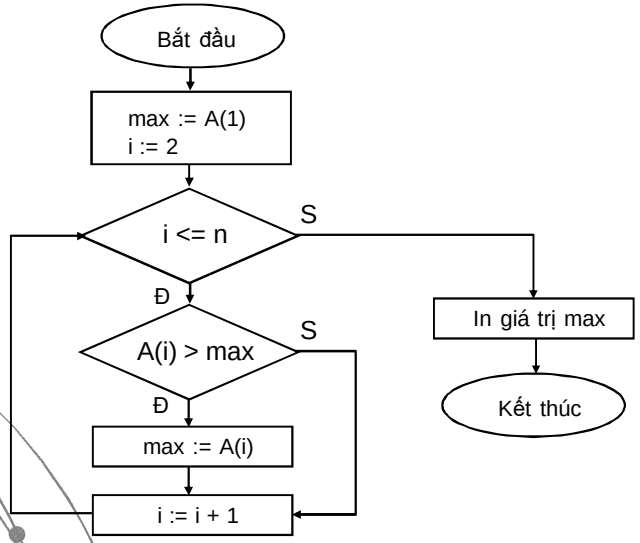


2.1. Lưu đồ giải thuật (*tiếp*)

- Các hình cơ bản trong lưu đồ giải thuật gồm có:
  - Hình thoi thể hiện các điệu kiện. Hình này có một đường vào và hai đường ra ứng với hai trường hợp điều kiện đúng hoặc điều kiện sai.



Ví dụ: Lưu đồ giải thuật tìm giá trị lớn nhất trong mảng số A có n phần tử



2.3. Giả mã

- Giả mã là giả ngôn ngữ lập trình (tựa ngôn ngữ lập trình).
- Trong cách diễn đạt giải thuật bằng giả mã, người ta sử dụng ngôn ngữ tự nhiên cùng với các cấu trúc chuẩn của một ngôn ngữ lập trình (Pascal) để mô tả giải thuật. Vì sử dụng cả ngôn ngữ tự nhiên nên có thể sử dụng các ký hiệu toán học để bản mô tả giải thuật ngắn gọn, dễ hiểu hơn.

2.3.1. Quy định chung

- Tên chương trình viết bằng chữ hoa, có thể thêm dấu gạch ngang và đặt sau từ Program
- Lời chú thích đặt giữa hai dấu ngoặc {….}. Lời chú thích được quy ước dùng tiếng Việt.

2.2.2. Biểu thức

- Các phép toán:
  - Số học: +, -, *, /, ^, DIV, MOD
  - Quan hệ: < , = , > , ≤ , ≥, ≠
  - Logic: NOT, AND, OR, XOR
  - Các giá trị Logic là True, False
- Tên biến là một dãy chữ cái, chữ số, dấu gạch nối (_), bắt đầu bằng chữ cái, độ dài không giới hạn.
- Biến chỉ số: Tên[chỉ số] Ví dụ : a[i], b[i,j]
- Biểu thức tương tự như Pascal.

2.2.3. Câu lệnh

- Các câu lệnh thể hiện các thao tác, công việc cần thực hiện. Các câu lệnh được viết viết cách nhau bởi dấu ;
- Phép toán gán được ký hiệu bởi dấu := hoặc ←
- Phép hoán đổi giá trị được ký hiệu bởi dấu :=: hoặc ↔
- Cấu trúc tuần tự: Liệt kê các công việc, các thao tác theo thứ tự. Để cho việc theo dõi được thuận tiện có thể đánh thêm thứ tự 1), 2), 3)… hoặc a), b), c)…
- Câu lệnh ghép:

  Begin s1; s2; ... ; sn; end

  Trong đó si là câu lệnh i

2.2.3. Câu lệnh

- Câu lệnh điều kiện:
  - if B then S;
  - if B then S1 else S2;

trong đó B là biểu thức logic, S là câu lệnh.

2.2.3. Câu lệnh

- Câu lệnh lựa chọn:

```
CASE
   B1: S1;
   B2: S2;
   . . .
   Bn: Sn;
   ELSE Sn+1;
END CASE
```

Với Bi (i=1, 2,…, n) là các điều kiện
Si (i=1, 2,…, n) là các câu lệnh

2.2.3. Câu lệnh

- Câu lệnh lặp:
  - Lặp với số lần lặp biết trước:

FOR i:=m TO n DO S;

FOR i:= n DOWNTO m DO S;

2.2.3. Câu lệnh

Lặp với số lần lặp không biết trước:

- Kiểm tra điều kiện trước:

WHILE B DO S;

- Kiểm tra điều kiện sau:

REPEAT S UNTIL B;

2.2.3. Câu lệnh

- Câu lệnh chuyển:

  GOTO n;

  trong đó n là số hiệu của một bước trong giải thuật.
- Câu lệnh vào ra:
  - READ(danh sách biến);
  - WRITE(danh sách hằng, biến, biểu thức);
- Câu lệnh kết thúc: END.

2.2.4. Chương trình con

2.2.4.1. Chương trình con dạng hàm

```
FUNCTION  Tên_hàm(danh sách tham số)
  S1; S2; . . .; Sn;
  Tên_hàm:= biểu thức;
RETURN
```

2.2.4.2. Chương trình con dạng thủ tục

```
PROCEDURE  Tên_thủ_tục(danh sách tham số)
  S1; S2; . . .; Sn;
RETURN
```


2.2.4.3. Lời gọi chương trình con

- Lời gọi chương trình con dạng hàm

  Tên_hàm(danh sách tham số thực sự)
- Lời gọi chương trình con dạng thủ tục

  CALL Tên_thủ_tục(danh sách tham số thực sự)

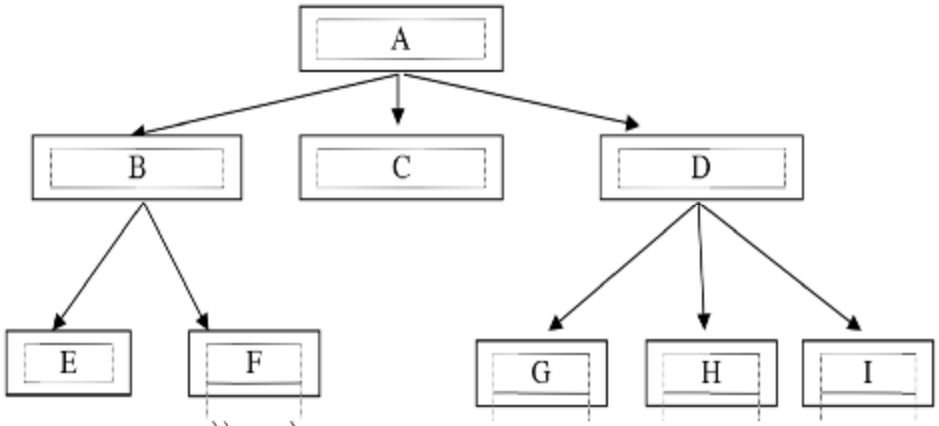
3. Thiết kế và phân tích giải thuật

3.1. Thiết kế thuật giải

3.1.1. Mô đun hóa việc giải quyết bài toán

- Khi thiết kế giải thuật ta sử dụng phương pháp mô đun hoá. Nội dung của phương pháp mô đun hoá là coi bài toán lớn như một mô đun chính và phân chia nó thành các mô đun con, mỗi mô đun con lại được phân chia tiếp, cho tới những mô đun ứng với các phần việc cơ bản mà ta đã biết cách giải quyết.
- Với phương pháp mô đun hoá bài toán thì lời giải của bài toán được tổ chức theo cấu trúc cây (phân cấp) có dạng như sau:

3.1.1. Mô đun hóa và việc giải quyết bài toán



3.1.1. Mô đun hóa việc giải quyết bài toán (*tiếp*)

- Chiến thuật giải quyết bài toán là chiến thuật " chia để trị", để thể hiện chiến thuật đó người ta dùng cách thiết kế "từ đỉnh xuống" (Top - Down).
- Cách thiết kế Top - Down hay thiết kế từ khái quát đến đến chi tiết thể hiện như sau: Phân tích tổng quát toàn bộ vấn đề xuất phát từ dữ liệu và mục tiêu đề ra, đề cập đến vấn đề chủ yếu, rồi sau đó mới đi dần vào giải quyết các vấn đề cụ thể một cách chi tiết hơn.

3.1.1. Mô đun hóa việc giải quyết bài toán (*tiếp*)

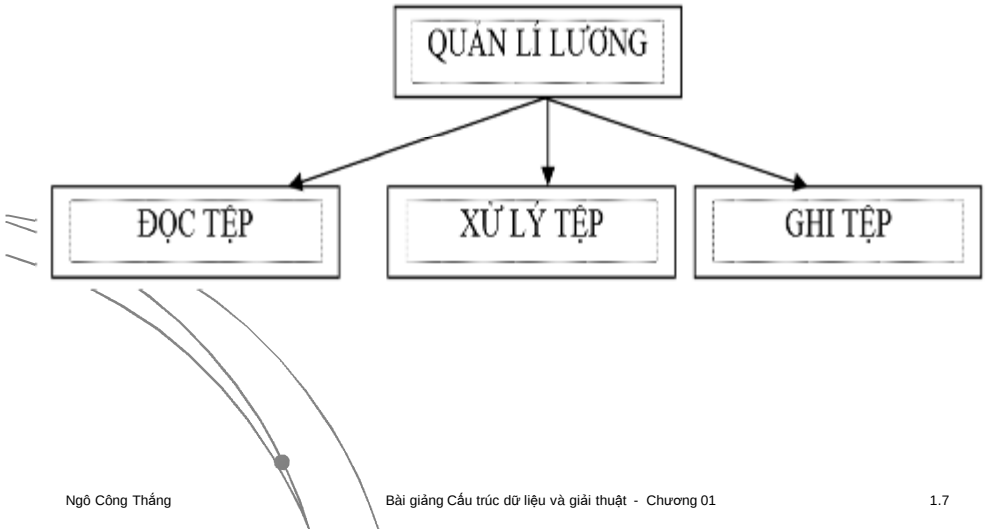
- Ví dụ: Bài toán đặt ra là dùng máy vi tính để quản lý lương của cán bộ trong xí nghiệp.
- Phân tích tổng quát bài toán:
  - Dữ liệu vào là một tệp hồ sơ về lương, bao gồm các bản ghi chứa các thông tin về lương của cán bộ. Bản ghi gồm các trường: mã, họ và tên, đơn vị, hệ số lương, phụ cấp, nợ.
  - Chương trình lập ra phải cho người sử dụng thực hiện được các công việc chính sau:
    1. Tìm kiếm thông tin
    2. Cập nhật thông tin
    3. In các bảng tổng hợp lương

3.1.1. Mô đun hóa việc giải quyết bài toán (*tiếp*)

- Xuất phát từ phân tích tổng quát ở trên ta thấy thuật giải xử lý phải giải quyết được 3 vấn đề sau:
  1. Đọc tệp: Đọc thông tin từ đĩa từ vào bộ nhớ
  2. Xử lý tệp: Xử lý các thông tin để đưa ra kết quả mong muốn
  3. Ghi tệp: Lưu trữ thông tin mới nhất vào tệp.

  Trên cơ sở này ta đưa ra sơ đồ giải thuật tổng quát như sau:

Sơ đồ giải thuật tổng quát

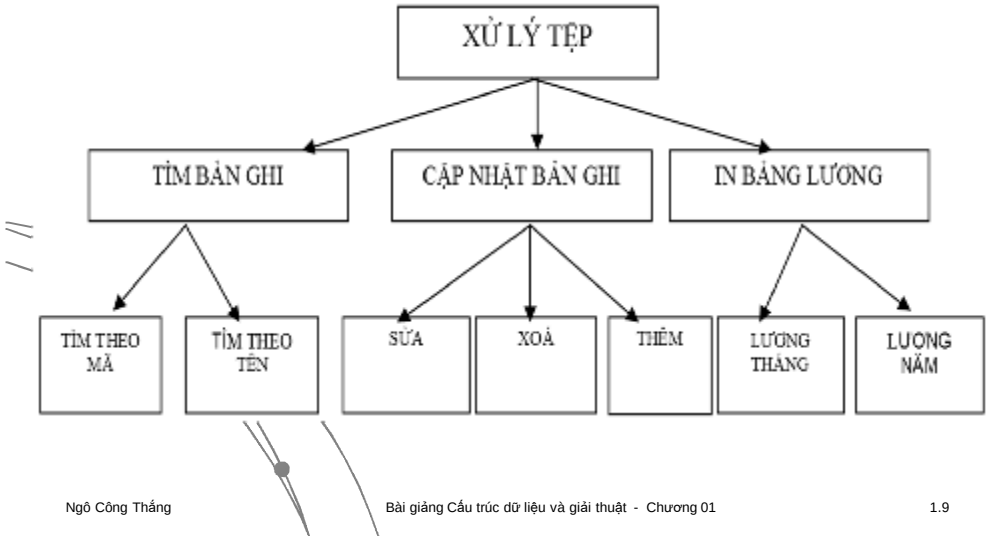


Ví dụ (*tiếp*)

- Các nhiệm vụ trên còn phức tạp, cần phải phân chia ra thành các nhiệm vụ con. Chẳng hạn nhiệm vụ " XỬ LÝ TỆP" được phân chia ra thành 3 nhiệm vụ con:
  1. Tìm kiếm bản ghi
  2. Cập nhật bản ghi
  3. In bản lương

  Những nhiệm vụ con lại được chia ra thành các nhiệm vụ nhỏ hơn theo sơ đồ sau:

Sơ đồ giải thuật chi tiết



3.1.1. Mô đun hóa việc giải quyết bài toán (*tiếp*)

- Ưu điểm của cách thiết kế Top - Down:
  - Giải quyết bài toán có định hướng, tránh sa đà vào các chi tiết phụ.
  - Làm nền tảng cho lập trình có cấu trúc.
  - Bài toán do nhiều người cùng làm, phương pháp mô đun hoá tách bài toán thành nhiều bài toán con tạo cho các nhóm làm việc độc lập, không ảnh hưởng đến nhóm khác.
  - Chương trình xây dựng trên giải thuật thiết kế theo kiểu Top - Down dễ dàng trong chỉnh sửa.

3.1.2. Phương pháp tinh chỉnh từng bước

- Phương pháp tinh chỉnh từng bước là phương pháp thiết kế giải thuật gắn liền với lập trình, nó phản ánh tinh thần của quá trình mô đun hóa bài toán và thiết kế kiểu Top - Down.
- Phương pháp này thể hiện như sau: Đầu tiên trình bày giải thuật bằng ngôn ngữ tự nhiên để phản ánh ý chính của công việc cần làm. Các bước tiếp theo sẽ chi tiết hoá dần dần, tương ứng với các công việc nhỏ hơn, gọi là các bước tinh chỉnh. Càng ở các bước sau thì công việc được mô tả hướng tới các lệnh của chương trình.

  Ngôn ngữ tự nhiên $\rightarrow$ Các bước tinh chỉnh $\rightarrow$ Giả mã

  Trong quá trình này dữ liệu cũng được tinh chỉnh dần từ dạng cấu trúc đến dạng cài đặt cụ thể.

Ví dụ 1: Sắp xếp một dãy n số nguyên theo thứ tự tăng dần

- Đầu tiên ta phác thảo giải thuật theo ngôn ngữ tự nhiên như sau:
  - Từ dãy các số nguyên chưa được sắp xếp lấy ra số nhỏ nhất.
  - Cứ lặp lại quá trình đó cho đến khi dãy chưa được sắp xếp trở thành rỗng.
- Các bước tinh chỉnh dùng giả ngôn ngữ Pascal là:

  1. Bước tinh chỉnh đầu tiên:

  For i:=1 To n-1 Do Begin
  - Xét từ a_i đến a_n để tìm số nhỏ nhất a_k
  - Đổi chỗ giữa a_i và a_k

  End

Ví dụ 1: Sắp xếp một dãy n số nguyên theo thứ tự tăng dần

- Các bước tinh chỉnh dùng giả ngôn ngữ Pascal là:

  2. Bước tinh chỉnh 2.1: Tìm số nhỏ nhất

```
k:=i
For j:= i+1 To n  Do
   If aj < ak  Then  k:=j
```

  3. Bước tinh chỉnh 2.2: Đổi chỗ

```
x:=ai ; ai:=ak ;  ak=x;
```


Ví dụ 1: Sắp xếp một dãy n số nguyên theo thứ tự tăng dần

- Sau khi chỉnh lại ta có thủ tục sắp xếp như sau:

```
Procedure SapXep(a,n)
    1) For i :=1 To n-1 Do
            Begin
    2)          k :=i
                For j :=i+1 To n Do
                    If a[j] < a[k] Then  k :=j
    3)          x:=a[i]; a[i]:=a[k]; a[k]:=x;
            End
Return
```

Ví dụ 2: Cho ma trận cấp mxn (m hàng, n cột). Tìm phần tử lớn nhất của các hàng và đổi chỗ nó cho phần tử đầu hàng.

- Phác hoạ thuật giải:
  1. Nhập m,n
  2. Nhập các phần tử của ma trận
  3. Tìm phần tử lớn nhất của các hàng và đổi chỗ cho phần tử đầu hàng.
  4. In ra ma trận

Ví dụ 2: Cho ma trận cấp mxn (m hàng, n cột). Tìm phần tử lớn nhất của các hàng và đổi chỗ nó cho phần tử đầu hàng.

- Diễn đạt bằng giả ngôn ngữ Pascal:

```
1. Read(m,n)
2. For i:= 1 To m  Do
        Đọc vào các phần tử hàng i
3. For i:= 1 To m Do
        Begin
          Tìm a[i,k] là phần tử lớn nhất cho hàng i
          Đổi chỗ giữa a[i,k] và a[i,1]
        End
4. For i:=1 To m Do
        In ra các phần tử hàng i
```


Ví dụ 2: Cho ma trận cấp mxn (m hàng, n cột). Tìm phần tử lớn nhất của các hàng và đổi chỗ nó cho phần tử đầu hàng

- Các bước được tinh chỉnh như sau:

```
1. Readln(m,n)
2. For i:=1 To m Do
        For j:= 1 To n Do
                Read(a[i,j])
3. For i:=1 To m Do
        Begin
                k:=1
                For j:=2 To n Do
                        If  a[i,j]> a[i,k] Then k:=j
                a[i,k]  <-> a[i,1]
        End
4. For i:= 1 To m Do
   Begin
        Writeln;
        For j:=1 To n Do
                Write(a[i,j], ‘  ‘)
   End
```


3.2. Phân tích, đánh giá giải thuật

3.2.1. Đặt vấn đề

- Phân tích tính đúng đắn: Chạy thử chương trình trên bộ dữ liệu, so sánh kết quả với kết quả đã biết.
- Các công cụ toán học chứng minh tính đúng đắn của giải thuật.
- Tính đơn giản: Dễ hiểu, dễ lập trình, dễ chỉnh lý.
- Phân tích thời gian: Thời gian thực hiện giải thuật là tiêu chuẩn đánh giá hiệu lực của giải thuật.

3.2.2. Phân tích thời gian thực hiện giải thuật

- Với một bài toán có nhiều giải thuật, ta cần chọn giải thuật dẫn đến kết quả nhanh nhất.
- Thời gian thực hiện phụ thuộc vào nhiều yếu tố như:
  - Kích thước của dữ liệu vào. Nếu gọi n là kích thước của dữ liệu 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, tốc độ xử lý của máy tính, ngôn ngữ viết chương trình, chương trình dịch cũng ảnh hưởng đến tốc độ thực hiện. Nhưng những yếu tố này không đồng đều với mỗi loại máy tính, vì vậy không thể đưa chúng vào xác lập T(n). Điều đó cũng có nghĩa là T(n) không thể tính theo đơn vị giây, phút…

3.2.3. Độ phức tạp tính toán của giải thuật

- Cách đánh giá thời gian thực hiện giải thuật không phụ thuộc vào máy tính và các yếu tố liên quan mà chỉ phụ thuộc vào kích thước dữ liệu đầu vào được gọi là đánh giá theo "Độ phức tạp tính toán của giải thuật".
- Nếu thời gian thực hiện một giải thuật là $T(n) = Cn^2$, trong đó C là hằng số, thì ta nói độ phức tạp tính toán của giải thuật này có cấp n^2, và được kí hiệu là: $T(n)= O(n^2)$
- Tổng quát: Hàm f(n) có độ phức tạp tính toán cấp g(n), kí hiệu là $f(n) = O(g(n))$, nếu tồn tại các hằng số C và n_0 sao cho:

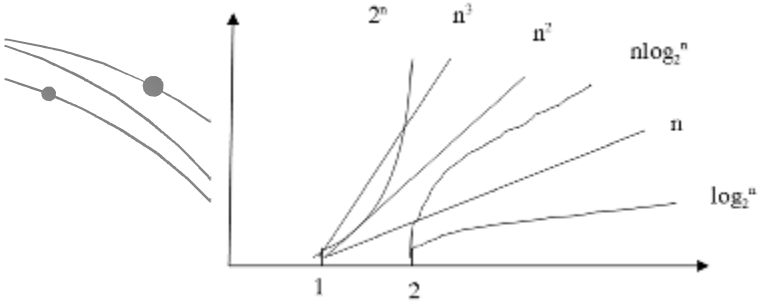
$$f(n) \le Cg(n) \text{ với } n \ge n_0$$

nghĩa là hàm f(n) bị chặn trên bởi Cg(n), với C là hằng số và với mọi n từ một điểm nào đó.

  - Ví dụ 1: $f(n) = O(n^3)$ có nghĩa độ phức tạp tính toán cấp n^3
  - Ví dụ 2: $f(n) = O(2^n)$ có nghĩa độ phức tạp tính toán cấp 2^n

3.2.3. Độ phức tạp tính toán của giải thuật (*tiếp*)

- Các hàm thể hiện độ phức tập tính toán của giải thuật có các dạng sau: n^n, n!, 2^n, n^3, n^2, $n\log_2 n$, n, $\log_2 n$. Các hàm này đã được sắp theo giá trị giảm dần, có nghĩa là với cùng một giá trị của n, hàm n^n là lớn nhất, $\log_2 n$ là nhỏ nhất. Các hàm này có dạng đồ thị như sau:



3.2.3. Độ phức tạp tính toán của giải thuật (*tiếp*)

- Các hàm n^n , n! , 2^n gọi là các hàm mũ. Một giải thuật có độ phức tạp tính toán cấp hàm mũ thì rất chậm, do đó khó được chấp nhận.
- Các hàm n^3, n^2, $n\log_2 n$, n, $\log_2 n$ là các hàm loại đa thức. Độ phức tạp tính toán của giải thuật có cấp đa thức thì chấp nhận được.

3.2.4. Xác định độ phức tạp tính toán

- Quy tắc cộng:
  - Giả sử T1(n) và T2(n) là thời gian thực hiện 2 đoạn chương trình P1 và P2 mà T1(n)= O(f(n)), T2(n)=O(g(n)), thì thời gian thực hiện P1 rồi đến P2 tiếp theo sẽ là: T1(n) + T2(n) = O(max(f(n),g(n)))
  - Ví dụ: Chương trình có 3 bước, mỗi bước có độ phức tạp tính toán lần lượt là $O(n^3)$, O(n), $O(n\log_2 n)$. Vậy thời gian thực hiện 3 bước là: T1(n) + T2(n) + T3(n) = O(max(n^3, n, $n\log_2 n$) = $O(n^3)$

3.2.4. Xác định độ phức tạp tính toán (*tiếp*)

- Quy tắc nhân:
  - Nếu tương ứng với 2 bước P1 và P2 là T1(n) = O(f(n)),T2(n) = O(g(n)) thì thời gian thực hiện P1 và P2 lồng nhau là : T1(n).T2(n) = O(f(n).g(n))
  - Ví dụ: Câu lệnh x:=x+1 có thời gian thực hiện bằng hằng số C => T(n) =O(1)

    Câu lệnh: For i :=1 To n Do x :=x+1;

    có thời gian thực hiện là: T(n)=O(n.1)=O(n)

    Câu lệnh

    ```
    For i :=1 To n Do
      For j :=1 To n Do x:=x+1;
    ```

    có thời gian thực hiện được đánh giá là: T(n)= O(n.n) = $O(n^2)$

3.2.4. Xác định độ phức tạp tính toán (*tiếp*)

- Quy tắc bỏ hằng số
  - O(c.f(n)) = O(f(n), trong đó c là một hằng số.
  - Ví dụ: $O(n^2/3) = O(n^2)$
- Chú ý 1: Khi đánh giá thời gian thực hiện giải thuật ta chỉ cần chú ý tới các bước tương ứng với một phép toán được gọi là phép toán tích cực. Đó là phép toán mà thời gian thực hiện nó không ít hơn thời gian thực hiện các phép toán khác.

3.2.4. Xác định độ phức tạp tính toán (*tiếp*)

- Ví dụ: $e^x = 1+ x/1! + x^2/2! + . . .+ x^n/n!$ với x và n cho trước.
- Giải thuật 1:

```
1) Read(x,n); s:=1;
2) For i :=1 To n Do begin
        p:=1;
        For j :=1 To i Do p:=p*x/j ;
       s:=s+p;
 end;
end.
```

3.2.4. Xác định độ phức tạp tính toán (*tiếp*)

- Trong giải thuật 1 phép toán tích cực ở đây là p:=p*x/j. Ta thấy nó được thực hiện với số lần là: 1+2+3+ . . . + n = n(n+1)/2

  Vậy thời gian thực hiện giải thuật là: T(n) = $O(n^2)$

3.2.4. Xác định độ phức tạp tính toán (*tiếp*)

- Giải thuật 2:

```
1) Read(x,n); s:=1; p:=1;
2) For i :=1  To  n  Do begin
        p:=p*x/i;
       s:=s+p;
   end;
end.
```

Thời gian thực hiện giải thuật 2 là: T(n) = O(n)

3.2.4. Xác định độ phức tạp tính toán (*tiếp*)

- Chú ý 2: Có những trường hợp thời gian thực hiện giải thuật không chỉ phụ thuộc vào kích thước của dữ liệu vào mà còn phụ thuộc vào chính tình trạng của dữ liệu đó nữa.
- Khi phân tích thời gian thực hiện giải thuật ta phải xét xem với mọi dữ liệu vào có kích thước n thì T(n) trong trường hợp thuận lợi nhất, trường hợp trung bình và trường hợp xấu nhất như thế nào?
- Việc xác định T(n) trong trường hợp trung bình thường khó vì phải dùng tới những công cụ toán đặc biệt. Bởi vậy người ta thường đánh giá giải thuật bằng T(n) trong trường hợp xấu nhất.

3.2.4. Xác định độ phức tạp tính toán (*tiếp*)

- Ví dụ: Cho véc tơ a có n phần tử a1, a2,..., an. Tìm trong a phần tử đầu tiên có giá trị = x cho trước.
- Giải thuật như sau:

```
Found := False;
i:=1;
While (i<=n) and Not Found  Do
  If   a[i] =x then  begin
       Found:=True;
       k:=i;
       Write(k);
  end
  else  i:=i+1;
End.
```

T(n) trong trường hợp tốt T(n)=O(1)

T(n) trong trường hợp xấu T(n)=O(n) . Vây suy ra T(n)=O(n)

4. Giải thuật đệ quy

4.1. Khái niệm đệ quy

- Ta nói một đối tượng là đệ quy nếu nó được định nghĩa dưới dạng chính nó.
- Ví dụ 1: Trên màn hình của vô tuyến truyền hình lại xuất hiện hình ảnh của chính cái màn hình vô tuyến đó.
- Ví dụ 2: Trong toán học hay gặp định nghĩa đệ quy:

1. Định nghĩa số tự nhiên
   a) x là số tự nhiên nếu x-1 là số tự nhiên
   b) 1 là số tự nhiên
2. Hàm n!
   a) n! = n*(n-1)! nếu n>0
   b) n!=1 nếu n=0

4.2. Giải thuật đệ quy và thủ tục đệ quy

- Nếu lời giải của một bài toán T được thực hiện bằng lời giải của bài toán T' có dạng giống như T thì đó là một lời giải đệ quy. Trong đó T' tuy giống T nhưng nó phải nhỏ hơn T.
- Giải thuật tương ứng với lời giải đệ quy gọi là giải thuật đệ quy.
- Thủ tục viết cho bài toán có lời giải đệ quy gọi là thủ tục đệ quy.

  Trong thủ tục đệ quy có lời gọi tới chính nó, mỗi lần gọi thì kích thước bài toán thu nhỏ hơn và dần dần tiến tới trường hợp đặc biệt là trường hợp suy biến.

Ví dụ: Bài toán tìm 1 từ trong cuốn từ điển

- Giải thuật đệ quy của bài toán này như sau:

  IF từ điển là một trang THEN tìm từ trong trang ấy

  ELSE BEGIN

  Mở từ điển vào trang giữa; Xác định xem nửa nào chứa từ

  IF từ nằm trong nửa trước THEN tìm trong nửa trước

  ELSE tìm trong nửa sau

  END

Ví dụ: Bài toán tìm 1 từ trong cuốn từ điển

- Trong giải thuật này có 2 điểm cần chú ý:
  - Điểm 1: Sau mỗi lần từ điển được tách đôi, một nửa thích hợp sẽ được tìm kiếm theo chiến thuật đã dùng.
  - Điểm 2: Có trường hợp đặc biệt là sau khi tách đôi từ điển chỉ còn 1 trang, giải quyết trực tiếp bằng cách tìm từ trong trang đó. Trường hợp đặc biệt này gọi là trường hợp suy biến.
- Giải thuật này gọi là giải thuật chia đôi: Bài toán được tách đôi ra bài toán nhỏ hơn, bài toán nhỏ hơn lại dùng chiến thuật chia đôi, cho tới khi gặp trường hợp suy biến.

Ví dụ: Bài toán tìm 1 từ trong cuốn từ điển

- Thủ tục đệ quy của bài toán được viết như sau:

```
Procedure  timkiem(Tudien, tu)
IF Tudien chỉ còn một trang   THEN tìm từ trong trang ấy
ELSE  BEGIN
    Mở từ điểm vào trang giữa
    Xác định xem nửa nào chứa từ
    IF Từ nằm ở nửa trước
    THEN  CALL timkiem(Tudien1, tu)
    ELSE   CALL timkiem(Tudien2, tu)
END
RETURN
```


4.3. Thiết kế giải thuật đệ quy

- Khi bài toán đang xét hoặc dữ liệu đang xử lý được định nghĩa dưới dạng đệ quy thì việc thiết kế các giải thuật đệ quy tỏ ra rất thuận lợi.
- Khi thiết kế giải thuật đệ quy cần trả lời các câu hỏi sau:
  1. Có thể định nghĩa bài toán dưới dạng một bài toán cùng loại nhưng nhỏ hơn như thế nào?
  2. Như thế nào là kích thước của bài toán được giảm đi ở mỗi lần gọi đệ quy?
  3. Trường hợp đặc biệt nào của bài toán sẽ được gọi là trường hợp suy biến?

Bài toán 1: Tính n!

- Định nghĩ đệ quy của hàm n! như sau:

  FAC(n) = 1 nếu n=0

  FAC(n)=n×FAC(n-1) nếu n>0

  Thuật giải đệ quy được viết dưới dạng hàm:

```
Function FAC(n)
  If  n=0  then  begin FAC :=1; return; end;
  Else  FAC := n * FAC(n-1);
Return
```


Bài toán 1: Tính n!

- Khử đệ quy của hàm tính giai thừa:

```
Function FAC(n)
    If  n=0  then  gt:=1;
    Else begin
                gt:=1;
                For i:=1 to n do gt:=gt*I;
          end;
    FAC:=gt;
Return
```

Bài toán 1: Tính n!

- Đối chiếu với 3 đặc điểm của thủ tục đệ quy ta thấy:
  - Lời gọi tới chính nó đứng sau Else
  - Mỗi lần gọi đệ quy giá trị giảm đi: FAC(4)→FAC(3)→FAC(2)→ FAC(1)
  - Trường hợp suy biến là FAC(0): FAC(0) = 1

Bài toán 2: Lập dãy số FIBONACCI
1 1 2 3 5 8 13...

- Định nghĩa F(n) như sau:

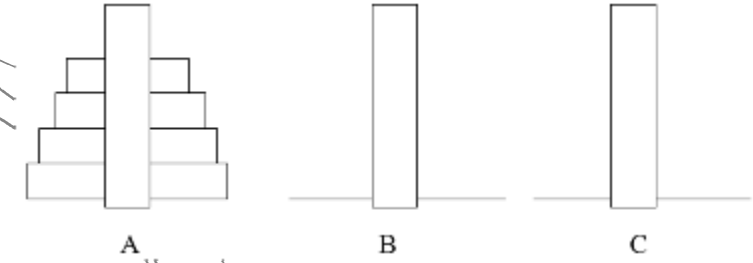
  F(n) = 1 nếu n ≤ 2

  F(n)=F(n-2)+F(n-1) nếu n>2
- Thủ tục đệ quy thể hiện giải thuật tính F(n) như sau:

```
Function  F(n:integer)
  If  n<=2 then   F:=1
   Else F:=F(n-2)+F(n-1)
Return
```


Bài toán 3: Bài toán "Tháp Hà nội"

- Nội dung: Có n đĩa kích thước nhỏ dần, đĩa có lỗ ở giữa. Có thể xếp chồng chúng lên nhau xuyên qua một cái cọc, to dưới nhỏ trên để cuối cùng có một chồng đĩa dạng như hình tháp.



Bài toán 3: Bài toán "Tháp Hà nội"

- Yêu cầu đặt ra: Chuyển chồng đĩa từ cọc A sang cọc C theo những điều kiện sau:
  - Mỗi lần chỉ được chuyển một đĩa
  - Không khi nào có tình huống đĩa to ở trên đĩa nhỏ ở dưới
  - Được phép sử dụng 1 cọc trung gian (cọc B) để đặt tạm thời.

Bài toán 3: Bài toán "Tháp Hà nội"

- Ta xét một vài trường hợp đơn giản:
  - Trường hợp 1 đĩa:
    - Chuyển từ cọc A sang cọc C
  - Trường hợp 2 đĩa:
    - Chuyển đĩa thứ nhất từ cọc A sang cọc B
    - Chuyển đĩa thứ hai từ cọc A sang cọc C
    - Chuyển đĩa thứ nhất từ cọc B sang cọc C

Bài toán 3: Bài toán "Tháp Hà nội"

- Trường hợp n đĩa (n>2): Ta coi n-1 đĩa ở trên như đĩa thứ nhất và xử lý giống như trường hợp 2 đĩa:
  - Chuyển n-1 đĩa từ cọc A sang cọc B
  - Chuyển đĩa thứ n từ cọc A sang cọc C
  - Chuyển n-1 đĩa từ cọc B sang cọc C
- Chuyển n-1 đĩa từ cọc B sang cọc C thuật giải sẽ là:
  - Chuyển n-2 đĩa từ cọc B sang cọc A
  - Chuyển 1 đĩa từ cọc B sang cọc C
  - Chuyển n-2 đĩa từ cọc B sang cọc C
- Cứ làm như vậy cho đến khi trường hợp suy biến xảy ra, đó là trường hợp ứng với bài toán chuyển 1 đĩa.

Bài toán 3: Bài toán "Tháp Hà nội"

- Thủ tục của bài toán "Tháp Hà nội" như sau:

```
Procedure  Hanoi(n,A,B,C)
If n=1 then chuyển đĩa từ A sang C
Else  Begin
      Call Hanoi(n-1,A,C,B)
      Call Hanoi(1,A,B,C)
      Call Hanoi(n-1,B,A,C)
      End;
Return
```


Bài tập

1. Thế nào là giải thuật đệ quy?
2. Ưu nhược điểm của giải thuật đệ quy?
3. Trong bộ nhớ của máy tính dùng vùng nhớ nào để dùng cho giải thuật đệ quy.
4. Trường hợp suy biến là trường hợp như thế nào trong giải thuật đệ quy.
5. Thường hay dùng cấu trúc lập trình nào để thể hiện giải thuật đệ quy
6. Viết giải thuật đệ quy cho bài toán sau:

   Acker(m,n)= n+1 nếu m=0

   Acker(m,n)= Acker(m-1,1) nếu n=0

   Acker(m,n)= Acker(m-1,Acker(m,n-1)) với các trường hợp khác.

Thủ tục đệ quy Bài 6

```
Function Acker(m,n:integer)
   If m=0 then Acker:=n+1
   else if n=0 then Acker:=Acker(m-1,1)
        else Acker:=Acker(m-1,Acker(m,n-1))
Return
```


Bài tập (tiếp)

7. Giải thuật tính ước số chung lớn nhất của hai số nguyên dương a và b (a>b) như sau: Gọi r là số dư của phép chia a cho b.
   - Nếu r=0 thì b là ước số chung lớn nhất
   - r khác 0 thì gán a:=b; b:=r rồi lặp lại.

   Hãy xây dựng giải thuật đệ quy tính ước số chung lớn nhất USCLN(a,b)

Thủ tục đệ quy bài 7

Bài toán: Tìm USCLN của 2 số nguyên dương a,b

Cách 1: Dùng đệ quy

```
Function USCLN(a,b)
   If b=0 then begin USCLN := a; return; end;
   If b # 0 then USCLN := USCLN(b,a mod b);
Return
```


Thủ tục đệ quy bài 7

Bài toán: Tìm USCLN của 2 số nguyên dương a,b

Cách 2: Khử đệ quy

```
Function USCLN(a,b)
   1) r := a mod b;
   2) while r # 0 do begin
            a:=b; b:=r; r:= a mod b
      end;
   3) USCLN:=b;
Return
```

Bài tập (tiếp)

8. Hàm C(n,k) với n, k là các giá trị nguyên không âm và k<=n, được định nghĩa như sau:

   C(n,n)=1

   C(n,0)=1

   C(n,k)=C(n-1,k-1)+C(n-1,k) nếu 0< k<n

   Hãy xây dựng giải thuật đệ quy cho bài toán trên

9. Viết thủ tục đệ quy in ngược một dòng kí tự cho trước.

Thủ tục đệ quy bài 8

```
Function C(n,k:integer)
   If k=0 then C:=1
   else if k=n then C:=1
            else C:=C(n-1,k-1)+C(n-1,k);
Return
```